UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Aplicación para crear chatbots y asistentes virtuales inteligentes

Máster Universitario en Ingeniería Informática

Autor: BUENO JIMÉNEZ, Adrián

Tutor: SÁNCHEZ-MONTAÑÉS ISLA, Manuel
Departamento de Ingeniería Informática

FECHA: Septiembre, 2019

Aplicación para crear chatbots y asistentes virtuales inteligentes

Bueno Jiménez, Adrián

Septiembre 2019

Resumen

Un chatbot es un programa informático capaz de comunicarse con una persona a través de un chat mediante el uso de lenguaje natural. Aunque los chatbots llevan existiendo casi desde los inicios de la computación, actualmente están generando mucho interés. Esto se debe a que la potencia de cómputo actual nos ha permitido aplicar algoritmos de aprendizaje automático y procesamiento de lenguaje natural más complejos y utilizar cantidades mayores de datos para generar mejores modelos. De esta forma obtenemos cada vez resultados más precisos en el entendimiento del lenguaje humano por parte de las máquinas. Estos avances tecnológicos han provocado que más empresas u organizaciones se interesen y entren en el mercado de los chatbots, y ofrezcan servicios o aplicaciones para crearte el tuyo propio. En este trabajo se desarrolla una aplicación que también permite crear chatbots usando NLU (entendimiento del lenguaje natural) y un gestor de diálogo de estados finito que funciona con reglas. Una de las ventajas que ofrece esta aplicación es la posibilidad de crear chatbots multi-idioma.

Abstract

A chatbot is a computer program capable of communicating with a person through a chat using natural language. Although chatbots have existed almost since the beginning of computing, they are currently generating a lot of interest. The current computing power has allowed us to apply more complex machine learning and natural language processing algorithms and to use larger amounts of data to generate better models. Now machines obtain better and more precise results in human language understanding. These technological advances have increase the interest of more companies and organizations in chatbots. This companies now offer services or applications to create your own chatbot. In this work we develop an application that also allows us to create chatbots using NLU (Natural Language Understanding) and a finite state dialogue manager that works with rules. One of the advantages of this application is the possibility to create multi-language chatbots.

Palabras clave

Chatbot, asistente, virtual, aplicación, construir, generar, crear, NLU, NLP, procesamiento, lenguaje, natural, entendimiento, gestor, diálogo, chat, web

Keywords

Chatbot, assistant, virtual, application, build, generate, NLU, NLP, natural, language, processing, understanding, dialog, manager, chat, web

Índice

1. INTRODUCCIÓN

1.1 Motivación

Un chatbot, o bot conversacional, es un programa informático que interactúa con personas a través de un chat textual o mediante voz y que ofrece respuestas y soluciones rápidas a tareas repetitivas y preguntas comunes susceptibles de ser automatizadas [15, 16].

Aunque los chatbots llevan existiendo casi desde los inicios de la computación [10], los grandes avances en inteligencia artificial de estos últimos años, han hecho resurgir el interés en estos; entre otras cosas gracias a las nuevas técnicas de reconocimiento y análisis del lenguaje natural [1, 14].

Los gigantes tecnológicos conocidos por todos, Google, Amazon, IBM, Microsoft, están invirtiendo muchos recursos en inteligencia artificial y machine learning. Todos ellos tienen una o varias herramientas y servicios para crear chatbots (Dialogflow, Amazon Lex, Watson, LUIS) [2]. Además de las grandes empresas, hay otra multitud de empresas más pequeñas que han creado plataformas y servicios enfocados exclusivamente a chatbots. Bastantes de estos están siendo usados en estrategias de marketing (Chatfuel, ManyChat, FlowXO, entre muchas otras) [3].

También se puede ver la apuesta por estas tecnologías en los asistentes personales como Alexa, Google y Siri, que podemos encontrar tanto en nuestros smartphones como en los altavoces inteligentes, con los cuales se puede interactuar mediante lenguaje natural haciendo uso de la voz [5]. Este es el ejemplo más claro que todos podemos percibir a diario [11]. Cuando te adentras un poco en el mundo de los chatbots, uno se da cuenta de la cantidad de empresas que están apostando por ellos y cómo está creciendo cada vez más el interés en estas tecnologías dentro de las mismas [6]. Además, podría decirse que los chatbots y los asistentes virtuales todavía están en su infancia, por lo que es un es un buen momento para hacerse un hueco en este mercado.

Algunas de las razones por las que los chatbots están ganando alta popularidad se debe a que no es necesario descargarse ninguna aplicación específica, lo habitual es que funcionen dentro de las aplicaciones de mensajería más conocidas. Se puede utilizar más de un chatbot en la misma aplicación de mensajería, están siempre disponibles, la interacción con los servicios de atención al cliente se automatizan y se puede dar respuesta a los usuarios de forma rápida para las preguntas y tareas más comunes sin que intervenga una persona. No obstante, cuando el bot no puede ofrecer la ayuda deseada, algunas plataformas de mensajería permiten pasar el control de la conversación a una persona. En definitiva, se mejora la experiencia de usuario porque se ofrecen soluciones de forma rápida y se utiliza una interfaz sencilla y conocida (un chat) [4, 7]. Otra razón importante por la que las empresas están apostando en los chatbots se debe a la reducción de costes producida por la mayor automatización de las tareas más comunes que a su vez ocasiona la reducción de trabajadores humanos que se encargaban de realizar estas [12, 13].

Existen bastantes aplicaciones para crear chatbots sin necesidad de programar una sola línea de código. Casi todas son de pago, aunque la mayoría de pago cuentan con una capa gratuita en la que no tiene coste su uso mientras que el chatbot tenga poco tráfico o tenga funcionalidades limitadas [2, 3]. También existen herramientas de software libre para crear chatbots, o una parte de ellos, tal como facilitar la conexión entre los servicios de mensajería y otros servicios [8] o realizar el reconocimiento de lenguaje natural [9]. Varias de estas herramientas y bibliotecas son mantenidas por empresas que siguen el modelo de: aplicación base de software libre y servicio de pago más avanzado y completo, con soporte. Por lo que a veces se echa en falta alguna funcionalidad más avanzada en la versión libre.

A lo largo de mi experiencia laboral he creado varios chatbots utilizando tanto servicios de pago para crear chatbots, como software libre para esto. También he creado funcionalidades de chatbots y análisis de lenguaje natural que no estaban disponibles en estos servicios, siguiendo los requisitos de los clientes. Según mi propia experiencia, las aplicaciones que permiten crear de forma sencilla y sin programar un chatbot son muy prácticas pero resulta complicado integrar servicios que no soportan y añadir nuevas funcionalidades. Además, están las limitaciones inherentes al software privativo. Al final siempre hace falta programar algún módulo porque cada cliente siempre solicita algún requisito distinto al resto. Además algunas empresas quieren que todo funcione dentro de su red y no quieren enviar datos que pueden ser sensibles a otra empresa. Pocos servicios ofrecen crear chatbots multi-idioma de forma fácil. Para uno de los chatbots desarrollados tuvimos que duplicar el diálogo existente, traducirlo y mantener las dos conversaciones por separado.

Mantener la misma conversación en varios idiomas se hace complicado. La mayoría de las aplicaciones para crear chatbots no tienen la posibilidad de tener varios entornos (ejemplo: desarrollo, producción). Algunos servicios en la nube de chatbots no tienen ninguna aplicación visual para crear el diálogo-flujo de la conversación, lo que hace difícil desarrollar conversaciones un poco más complejas.

El objetivo de este TFM es crear es un conjunto de aplicaciones que permitan desarrollar chatbots de forma sencilla y que solucionen los problemas descritos en el párrafo anterior. Se podrá crear nuevos módulos para extender la funcionalidad de la aplicación y será fácil compartirlos. La aplicación estará dirigida principalmente a desarrolladores y el código resultante del trabajo estará bajo una licencia de software libre, de esta forma cualquiera podrá adaptar la aplicación a sus necesidades.

1.2 Objetivos

El primer objetivo en este TFM es realizar un estudio extenso del estado del arte en el campo de bots conversacionales.

A continuación se diseñará y creará un conjunto de herramientas y aplicaciones que permitan crear chatbots y asistentes personales de forma rápida y sencilla.

Las aplicaciones que se van a diseñar y crear son:

- Servicio NLU (Natural Language Understanding). Esta aplicación analizará los mensajes para intentar extraer significado de ellos y dar respuestas coherentes a los usuarios). Será desarrollado en Python 3.
- Aplicación de diseño visual. Una aplicación visual permitirá un desarrollo más rápido y sencillo. Esta aplicación consistirá en una página web desarrollada con Angular y un servicio de backend con el que se comunicará a través de una API REST. Los chatbots diseñados podrán exportarse en formato JSON.
- Orquestador. Procesa los mensajes recibidos de los chats y utiliza todos los servicios necesarios para dar una respuesta a cada mensaje. Utilizará el fichero JSON generado por la aplicación visual para conocer los módulos que tiene que utilizar y el flujo de los datos. Estará integrado

con Facebook Messenger y Telegram. Soportará al menos mensajes de texto y botones de respuesta rápida. También se integrará con el servicio de NLU. El orquestador será desarrollado con Node.js.

Finalmente, el objetivo final es crear con las aplicaciones anteriores un chatbot de suficiente complejidad.

1.3 Organización del documento

En el apartado 2 se hace un breve resumen de la historia de los chatbots. A continuación se describen varias técnicas y herramientas utilizadas en la creación de chatbots. En el apartado 3 se encuentra la explicación del diseño de la aplicación creada. Posteriormente, en el apartado 4 se comentan los resultados obtenidos. En el último apartado, el 5, se realizan una conclusiones y se mencionan varias mejoras que se pueden realizar a futuro.

2. ESTADO DEL ARTE

2.1 Breve historia de los chatbots

En el año 1950 Alan Turing publicó el paper "Computer Machinery and Intelligence" donde definió una prueba, conocida como el *Test de Turing*, para comprobar si una máquina puede llegar a tener un comportamiento inteligente similar o indistinguible al de un ser humano. La prueba consiste en hacer conversar a través de un chat, a una persona con otra entidad, que puede ser otra persona o una máquina. Esta persona tiene que determinar si con quién está hablando es una máquina o no [18].

El chatbot ELIZA, creado en 1966 por Joseph Weizenbaum, es considerado como el primer software conversacional. El objetivo de ELIZA era emular a un psicoterapeuta. ELIZA conversaba de forma escrita con su interlocutor y tenía que aparentar que escuchaba y entendía sus problemas. Para conseguir un diálogo más humano, ELIZA hacía uso de reglas, reconocimiento de patrones y detección de palabras clave que luego utilizaba para realizar preguntas o responder con frases hechas, empáticas y de continuidad que ya tenía almacenadas. De esta forma la persona tenía la sensación de que lo que estaba al otro lado del chat estaba mostrando interés y continuaba hablando. Cuando no tenía respuestas disponibles, hacía uso de otra técnicas, como generar preguntas utilizando como base la frase que uno le decía [19].

El sucesor de ELIZA fue PARRY, creado por Kenneth Colby en 1971. Utilizaba una arquitectura similar y simulaba ser una paciente con paranoia. A diferencia de ELIZA, PARRY si tenía conocimiento de la conversación y también de su estado de ánimo. Las respuestas no venían determinadas sólo por la entrada, sino también por las creencias, deseos e intenciones de PARRY. Algunos trucos que utilizaba eran admitir ignorancia, cambiar el tema de conversación e introducir durante la conversación pequeñas historias de la mafia [22].

Otro chatbot representativo fue A.L.I.C.E. (Artificial Linguistic Internet Computer Entity), creado en 1995 por Richard Wallace. Este bot participó en

numerosos concursos relacionados con la evaluación del procesamiento del lenguaje natural y obtuvo muchos honores y premios. Ganó el premio Loebner 3 veces. [20]. Utiliza un reconocimiento de patrones avanzado, AIML (Artificial Intelligence Markup Language), desarrollado también por Wallace [21].

El Premio Loebner es una competición anual que sigue el estándar establecido por Alan Turing. Fue iniciado en 1991 por Hugh Loebner y el Centro de Estudios de la Conducta de Cambridge. Concede premios a los chatbots que estén considerados por el jurado como los más inteligentes entre todos los que se presentan. Por ahora ningún chatbot ha sido capaz de superar el test de Turing. [25, 26]. Existe cierta controversia con este concurso debido a que los chatbots intentan fingir ser humanos pero no llegan a tener una inteligencia real. Algunos chatbots cometen faltas de ortografía, entre otras técnicas, para parecer más humanos [17].

Albert One, desarrollado por Robby Garner en 1997, fue ganador del premio Loebner en 1998 y 1999. Se volvió más popular por su comportamiento humano. Albert figura en el Libro Guinness de los Récords Mundiales de 2001 como el programa informático más humano del mundo. [23] Antes de Albert One, Garner creó a FRED, un programa que empezaba con un conjunto básico de frases y aprendía de otras conversaciones para generar respuestas. El problema de este tipo de aprendizaje no supervisado, es que se pueden obtener resultados no deseables [22], como fue el caso del chatbot de Microsoft, Tay, que utilizó Twitter para aprender de los tweets que le mandaban los usuarios de la plataforma y se volvió racista en menos de un día (2016) [24].

El último chatbot en ganar el premio Loebner ha sido Mitsuku. Ha conseguido el premio cuatro veces: 2013, 2016, 2017 y 2018. [27, 29] Fue desarrollado por Steve Worswick utilizando como base los ficheros AIML de A.L.I.C.E. Tiene una base de datos de palabras de muchos objetos comunes diferentes. Cada objeto tiene muchas propiedades que lo describen, por lo que es fácil para Mitsuku razonar sobre ellos sin que haga falta codificar miles de respuestas [28].

2.2 Técnicas para construir chatbots de manera más “manual”

Antes del boom de las técnicas de aprendizaje automático el desarrollo de chatbots consistía principalmente en especificar reglas con las cuales se generaría la respuesta. Estas reglas pueden ir desde la simple coincidencia de patrones hasta la inclusión de la estructura gramatical de la oración. Actualmente, para la construcción de agentes conversacionales suele utilizarse AIML [31] (descrita en el apartado 2.3) y NLP/NLU (Natural Language Processing / Natural Language Understanding) [30].

2.2.1 Identificación de patrones

La identificación de patrones han sido la técnica más usada para construir chatbots [10, 30]. Con esta técnica se identifica la estructura de la oración y se devuelve una respuesta predefinida según las variables características de la oración [30] y palabras clave [10]. Por ejemplo, cuando ELIZA analizaba una frase como: “Dice que estoy deprimido la mayor parte del tiempo”, aplica una regla para transformar “estoy” por “estás”. El algoritmo de generación añade “Siento oír que” a la frase y se devuelve “Siento oír que estás deprimido” [10]. Esta metodología es más típica de chatbots de tipo pregunta-respuesta. La principal desventaja de este enfoque es que las conversaciones se vuelven repetitivas y poco humanas [30].

2.2.2 Análisis sintáctico

Con el análisis léxico se convierte el texto original en un conjunto de palabras con características (nombre, adjetivo, verbo, etc.) para determinar su estructura gramatical. Con el análisis sintáctico se aplican las reglas de la gramática del idioma del texto y se determina el papel de cada palabra en la oración. De esta forma se puede comprobar si el texto tiene una estructura gramatical correcta [17, 34]. Los primeros analizadores de texto eran muy simples y solo comprobaban que las palabras clave estuvieran en el orden correcto. Con este enfoque se podía cubrir múltiples frases de entrada con un conjunto más limitado de patrones, por ejemplo, las frases “Puedes traerme un poco de agua” y “Me podrías dar un vaso de agua” se convierten en “dar agua” [17].

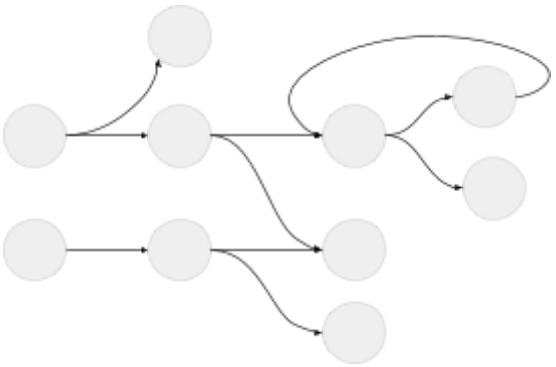
2.2.3 Ontologías (redes semánticas)

Una ontología es una descripción formal explícita de clases o conceptos, propiedades de cada concepto que describen características y atributos del concepto (ranuras) y restricciones de las ranuras. Una ontología junto con un conjunto de clases, forman una base de conocimiento. Las clases describen conceptos en el dominio. Por ejemplo, una clase de vinos representa todos los vinos. Los vinos específicos son ejemplos de esta clase. Una clase puede contener a otras subclases que representan conceptos más específicos que la superclase. Por ejemplo, podemos dividir la clase de vinos en las subclases de tintos, blancos y rosados. A nivel de clase, se puede decir que las instancias de la clase vino tendrán propiedades que describen su sabor, color, cuerpo, etc. El desarrollo de una ontología incluye: definir las clases, organizar las clases en una jerarquía taxonómica (subclase-superclase), definir las ranuras y describir los valores permitidos para estas ranuras, y completar las ranuras con valores para cada instancia de la clase [37].

La ventaja de utilizar una ontología como base de conocimiento es que los sistemas pueden buscar a través de los nodos del gráfico de conocimiento para establecer relaciones entre los conceptos que se utilizan en la interacción y también implican un nuevo razonamiento. Wordnet y OpenCyc son ontologías populares [30].

2.2.4 Gestor de diálogo

Un gestor de diálogo se encarga de gestionar y controlar el estado del flujo de la conversación. El tipo de gestor más típico está basado en los sistemas de estados finitos. El flujo del diálogo se especifica a través de una conjunto de estados con transiciones que denotan varias rutas alternativas a través de un gráfico de diálogo. En cada estado el sistema realiza distintas acciones dependiendo de la entrada reconocida. Los estados de diálogo y las transiciones deben diseñarse de antemano. [10]



Ejemplo de gestor basado en sistema de estados finitos. Cada círculo representa un estado y cada flecha la transición de un estado a otro.

2.3 Técnicas estadísticas y Machine Learning

2.3.1 Cadenas de Markov

Una cadena de Markov es un modelo probabilístico en el que la probabilidad de que ocurra un evento depende únicamente del evento anterior. Con este método lo que se pretende conseguir es modelar las probabilidades de las transiciones de estado a lo largo del tiempo. La idea principal es que existe una probabilidad fija de que ocurra la misma letra en los mismos datos textuales [30].

En los chatbots se utilizaron los modelos de cadenas de Markov para construir respuestas que son probabilísticamente más viables y, por lo tanto, más correctas. En algunos casos (HeX) se utilizaron para generar una frase sin sentido pero que suena bien, cuando el chatbot se queda sin respuestas [17].

También existen otros usos, por ejemplo, se puede utilizar para autocompletar consultas, prediciendo términos de búsqueda que coincidan con el contenido de algún documento, sugiriendo al usuario palabras que puede emplear en la búsqueda [38].

2.3.2 Recuperación de información

La recuperación de información, Information Retrieval en inglés (IR), consiste en buscar material, normalmente documentos, de naturaleza no estructurada, generalmente texto, que satisface una necesidad de información dentro de grandes colecciones, generalmente almacenadas en computadoras [45]. Los mejores ejemplos de sistemas de recuperación son los buscadores web, como Google, Yahoo o Bing. [81]

Un chatbot puede utilizar esta técnica de búsqueda para recuperar la pregunta más cercana en una base de conocimiento de preguntas y respuestas y devolver la respuesta asociada [46].

2.3.3 Redes Neuronales

Las redes neuronales son un modelo computacional inspirado por el comportamiento de las neuronas biológicas. Una red neuronal está formada por un conjunto de unidades, llamadas neuronas, interconectadas entre sí. Cada neurona recibe como entrada un conjunto de señales discretas o continuas, las pondera e integra, y transmite el resultado a las neuronas conectadas a ella. Cada conexión entre dos neuronas tiene una determinada importancia asociada denominada peso. La mayor parte del conocimiento que la red tiene sobre una tarea en concreto se guarda en los pesos. Mediante el entrenamiento de la red se ajustan estos pesos para lograr una determinado objetivo [39].

Muchos trabajos recientes han demostrado que las redes neuronales pueden utilizarse con éxito en una serie de tareas de procesamiento del lenguaje natural (NLP). Estos incluyen, pero no se limitan a, modelado de lenguaje, detección de paráfrasis, extracción de embeddings de palabras y reconocimiento de voz [48].

El tipo de red neuronal más utilizado para el procesamiento de lenguaje natural es la Red Neuronal Recurrente (RNN). Este tipo de red incorpora retroalimentación, lo que permite que la red tenga memoria. La ventaja de estas redes es que pueden generalizar a través de secuencias en lugar de aprender patrones individuales. Tampoco están limitadas a un tamaño fijo de secuencia, y en teoría, pueden tener en cuenta todos los pasos previos de la secuencia. Se utilizan para procesar datos secuenciales [40].

Con las arquitecturas de capas LSTM (Long Short-Term Memory, memoria a largo plazo) o capas GRU (Gated Recurrent Unit, unidad recurrente cerrada) podemos crear redes recurrentes profundas (DRNN). Estas arquitecturas permiten recordar información por largos periodos de tiempo [43].

También pueden utilizarse otro tipos de redes, como las redes convolucionales (CNN). Son un tipo de red en donde las neuronas corresponden a campos receptivos inspiradas por las neuronas de la corteza visual primaria [41]. Esta arquitectura permite procesar entradas de datos en una dimensión (por ejemplo, lenguaje natural o series temporales), dos dimensiones (por ejemplo, imágenes en gris), tres dimensiones (por ejemplo, imágenes a color o imágenes 3D del cerebro en imagen médica), o incluso más dimensiones [42].
[42].

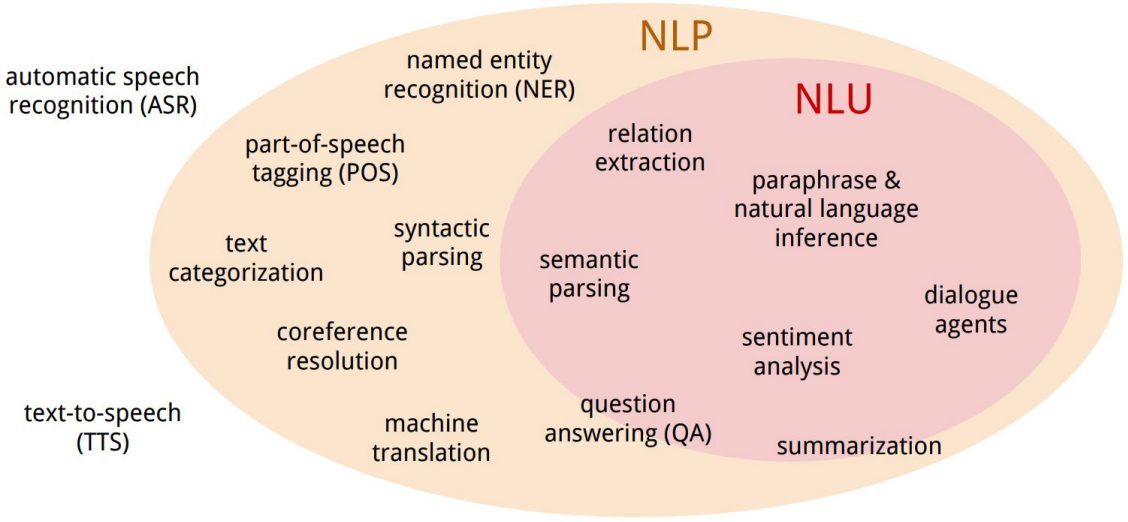
Los modelos de secuencia a secuencia (seq2seq) se pueden construir con RNNs o DRNNs. Los modelos de secuencia a secuencia aprenden a generar una secuencia de longitud variable de tokens, por ejemplo, textos, a partir de una secuencia de longitud variable de datos de entrada, por ejemplo, el habla (audio) o textos. Han logrado obtener resultados de vanguardia en muchos problemas de procesamiento de lenguaje natural, incluyendo el reconocimiento del habla, la traducción neural automática, el modelado conversacional y muchos más [44]. Estos modelos se pueden utilizar también en los chatbots donde la secuencia de entrada sería el mensaje de texto del usuario y la secuencia de salida sería el mensaje de respuesta [30].

2.3.4 NLP/NLU

El procesamiento del lenguaje natural (NLP en inglés) es una rama de la inteligencia artificial que ayuda a las computadoras a entender, interpretar y manipular el lenguaje humano. Toma elementos prestados de otras disciplinas, como la ciencia de la computación y la lingüística computacional. NLP es importante porque ayuda a resolver la ambigüedad del lenguaje y agrega estructura numérica útil a los datos para muchas aplicaciones industriales, como el reconocimiento del habla o la analítica de texto [47].

El NLP utiliza algunas de las técnicas descritas más arriba para realizar tareas como: extracción de información, similitud, reconocimiento del habla, generación de lenguaje natural y voz, resumen de textos, análisis de sentimientos, extracción de temas, reconocimiento de entidades con nombre, extracción de relaciones, minería de textos, traducción, respuesta automática a preguntas [48].

A veces cuando se habla de NLP también se habla de NLU (Comprensión del Lenguaje Natural). NLU es un subconjunto de NLP que engloba métodos de más alto nivel enfocados, como su nombre indica, en la comprensión del lenguaje.



NLP y NLU (2014) [53]

El NLP/NLU es usado en los chatbots para extraer de los mensajes de los usuarios las intenciones y las entidades. Las intenciones (intents en inglés) corresponden a qué acciones deben realizarse como respuesta a una entrada del usuario. Las intenciones están formadas por un conjunto de frases de ejemplo (llamadas utterances en inglés) con estructura distinta pero con el mismo significado (por ejemplo, "Pon música", "Quiero escuchar mis canciones favoritas", "Reproduce mis canciones" estarían en una intención con nombre "ReproducirMúsica"). Las entidades son variables del lenguaje natural que están asociadas a ciertas palabras o conjunto de palabras. Algunos ejemplos de entidades: fechas y horas (5 de septiembre", "10:15", "lunes"), nombres de personas ("Pedro", "María"), lugares ("Madrid", "Barcelona"), tipos de animales ("gato", "perro"), etc. Los slot sirven para dar nombres a las entidades de una frase para poder, por ejemplo, distinguir varias entidades del mismo tipo. En la frase: "¿Cómo puedo ir de Madrid a Barcelona utilizando el transporte público?" Madrid y Barcelona pertenecen a la misma entidad "Localización", pero Madrid puede asociarse a un slot "LocalizaciónOrigen" y Barcelona a un slot "LocalizaciónDestino".

2.4 Preprocesado

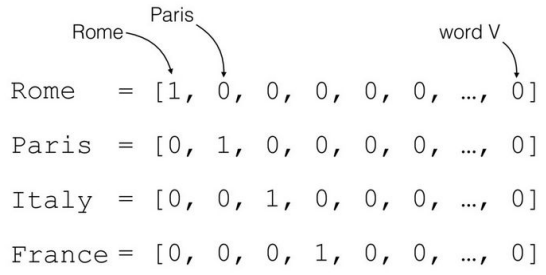
Para obtener mejores resultados en los técnicas estadísticas, los textos suelen ser preprocesados. Las principales técnicas de preprocesado de texto son: segmentación de textos en frases, tokenización (dividir frases en palabras), stemming (reducir palabra a su raíz), lematización (hallar el lema de la palabra), etiquetado de partes del discurso (análisis sintáctico), eliminación de stopwords (palabras muy comunes y sin contenido semántico, como “a”, “y”, “como”, “en”, “la”, “para”, etc.), convertir todas las palabras a minúsculas o mayúsculas, normalizar la ortografía para eliminar faltas de ortografía [49].

Además las máquinas no son capaces de entender las palabras tal cual, es necesario utilizar una codificación numérica. Para ello se convierten las palabras a una representación vectorial.

2.4.1 Codificación vectorial

2.4.1.1 Bolsa de palabras

Una de las técnicas más simples de codificación vectorial es Bolsa de Palabras (BOW) o Codificación One-hot donde el tamaño de los vectores es igual al número total de palabras únicas, y donde todos los valores del vector valen 0 excepto la posición que corresponde al índice de cada palabra. Se obtienen vectores enormes sin ninguna relación semántica [50].



Ejemplo de Bolsa de Palabras [50]

2.4.1.2 TF-IDF

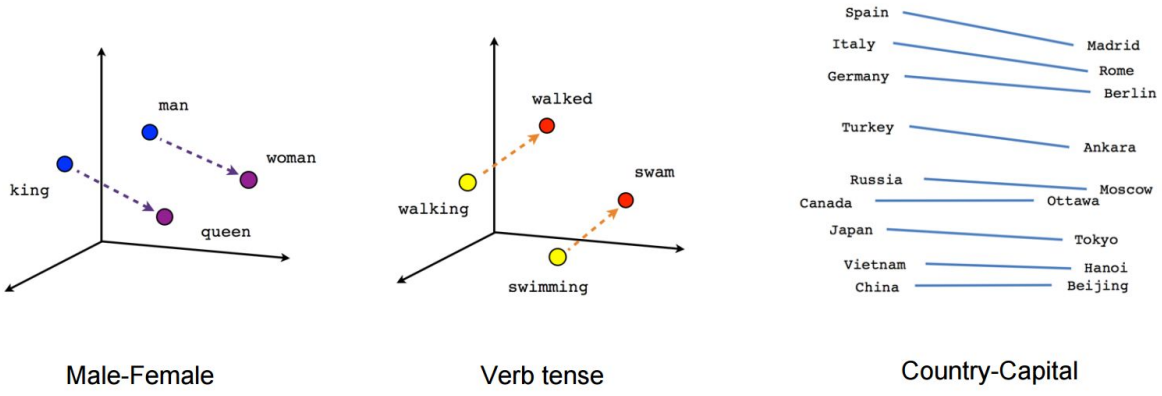
Otro método muy usado es TF-IDF (Term Frequency - Inverse Document Frequency), donde los vectores están relacionados con los vectores de la bolsa de palabras, sin embargo, las palabras se representan por su frecuencia de término multiplicada por su frecuencia de documento inversa, es decir, las palabras que aparecen mucho pero en todas partes deben tener muy poca ponderación o importancia, y las palabra que aparecen poco o con frecuencia pero en pocos documentos, se consideran más importantes [50]. Este método es muy usado en los sistemas de recuperación de información.

2.4.1.3 LSA

El análisis semántico latente (LSA) es un método que extrae y representa el significado de uso contextual de las palabras mediante cálculos estadísticos aplicados a un gran corpus de texto. La idea es que el conjunto de todos los contextos de palabras en los que aparece y no aparece una palabra, proporcionan un conjunto de restricciones que determinan en gran medida la similitud del significado de las palabras y los conjuntos de palabras entre sí [52].

2.4.1.4 Word2Vec

Actualmente se utilizan también las redes neuronales para obtener vectores de palabras con relaciones entre ellos. Uno de los modelos más conocidos es word2vec. Se encuentra disponible de dos formas: el modelo continuo de bolsa de palabras (CBOW) y el modelo Skip-Gram. Algorítmicamente, estos modelos son similares, excepto que CBOW predice las palabras objetivo (por ejemplo, “alfombra”) a partir de las palabras de contexto fuente (“el gato se sienta en la”), mientras que el modelo skip-gram hace lo contrario y predice palabras contextuales de la fuente a partir de las palabras objetivo. Normalmente CBOW funciona mejor en conjuntos de datos pequeños y Skip-Gram en conjuntos grandes. La ventaja de este modelo es que los vectores capturan información semántica general sobre las palabras y sus relaciones entre sí, como se puede observar en la imagen siguiente [51]:



Relaciones entre palabras obtenida con word2vec [51]

Estas relaciones son útiles para tareas de NLP como el etiquetado de partes del discurso (part-of-speech tagging) o el reconocimiento de entidades con nombre [51].

2.5 Otras herramientas

AIML, Cleverscript y Chatscript son herramientas que te dan todas las funcionalidades necesarias para construir un chatbot, pues permiten desde entender las preguntas del usuario hasta generar u obtener una respuesta. Estas herramientas utilizan también algunas de las técnicas mencionadas, como la identificación de patrones.

2.5.1 AIML

AIML (Artificial Intelligence Markup Language) es un dialecto de XML. Se diseñó a finales de los años 90 con la explosión de la World Wide Web. Se quería que AIML fuera igual de sencillo que HTML. A la vez, en esos años apareció XML y los autores de AIML decidieron utilizar muchas herramientas desarrolladas para XML. Aunque AIML está vinculado al XML, no depende de él, mientras que se use la misma estructura de fichero se puede usar JSON, YAML o incluso expresiones S de LISP [32].

AIML describe una clase de objetos llamados objetos AIML. También describe parcialmente el comportamiento de los programas informáticos. Los programas AIML están formados por unidades llamadas temas `<topic>` y categorías `<category>`, que contienen datos parseados o no parseados [31].

Las categorías son la unidad básica de AIML. Cada categoría consta de una pregunta de entrada, una respuesta de salida y un contexto opcional. La pregunta se llama patrón `<pattern>` y la respuesta plantilla `<template>`. Los dos tipos principales de contexto se llaman "eso" `<that>` y tema `<topic>` [31].

El lenguaje de patrones de AIML es simple, consiste solo de palabras (con sólo caracteres alfanuméricos), espacios y símbolos comodín como _ y *. Las palabras se separan por un solo espacio y los caracteres comodín funcionan como palabras. No se distingue entre mayúsculas y minúsculas [31].

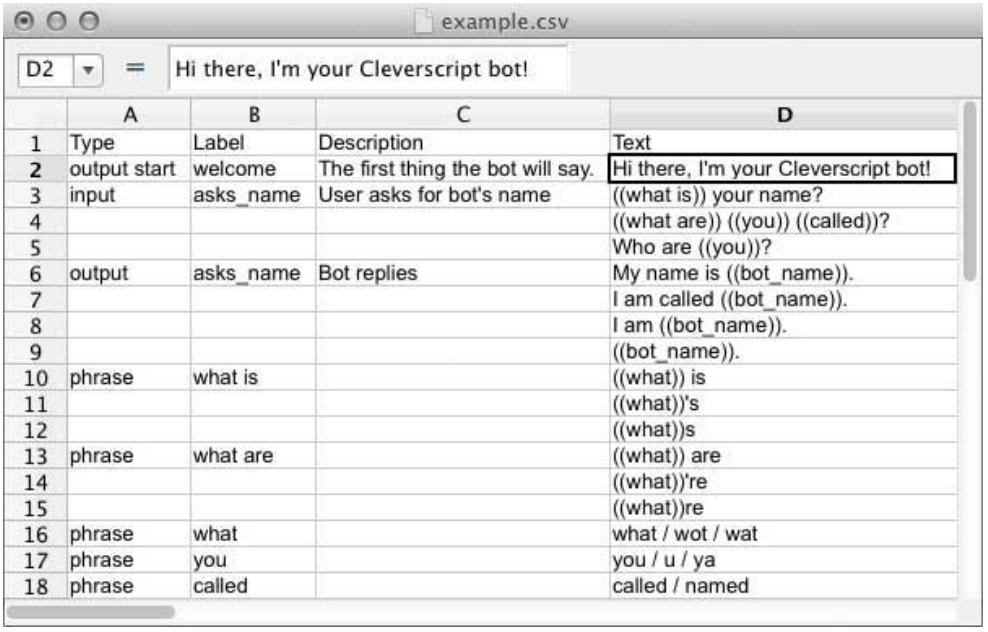
Las etiquetas AIML transforman la respuesta en un mini programa informático que puede guardar datos, activar otros programas, dar respuestas condicionales y llamar recursivamente al buscador de patrones para insertar las respuestas de otras categorías [31].

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">
   <category>
       <pattern>WHAT IS MY NAME</pattern>
       <template><get name="name" /></template>
   </category>
   <category>
       <pattern>MY NAME IS *</pattern>
       <template>Nice to meet you, <set name="name"><star /></template>
   </category>
   <category>
       <pattern>DO YOU FIND ME ATTRACTIVE</pattern>
       <template>
       <condition name="gender">
           <li value="male">I find you very handsome.</li>
           <li value="female">I find you very pretty.</li>
           <li>I find you very attractive.</li>
       </condition>
       </template>
   </category>
</aiml>
```

Ejemplo sencillo de programa AIML para la versión 2.0 [33]

2.5.2 Cleverscript

Es un motor de inteligencia artificial y comprensión del lenguaje natural. Se utiliza para analizar texto o voz en datos estructurados. Los bots de Cleverscript se crean usando hojas de calculo. La idea básica es escribir entradas (cosas que puede reconocer) y salidas (respuestas que puede dar) en líneas separadas de la hoja de cálculo. Las entradas pueden tener anotaciones especiales como los dobles paréntesis ((<palabra>)) para definir un conjunto de posibles palabras con significados iguales o similares. Por ejemplo, en las frase "((querer)) comer un helado", ((querer)) puede contener los valores "quiero", "me apetece", "deseo", etc. Los errores tipográficos y ortográficos también están permitidos [35].



	A	B	C	D
1	Type	Label	Description	Text
2	output start	welcome	The first thing the bot will say.	Hi there, I'm your Cleverscript bot!
3	input	asks_name	User asks for bot's name	((what is)) your name?
4				((what are)) ((you)) ((called))?
5				Who are ((you))?
6	output	asks_name	Bot replies	My name is ((bot_name)).
7				I am called ((bot_name)).
8				I am ((bot_name)).
9				((bot_name)).
10	phrase	what is		((what)) is
11				((what))'s
12				((what))s
13	phrase	what are		((what)) are
14				((what))'re
15				((what))re
16	phrase	what		what / wot / wat
17	phrase	you		you / u / ya
18	phrase	called		called / named

Ejemplo de hoja de cálculo con el formato de Cleverscript [35]

También permite utilizar variables para poder devolver distintas respuestas dependiendo del valor de estas o para recordar datos hasta que la consulta del usuario se resuelva [35].

2.5.3 Chatscript

Chatscript es un motor basado en reglas. Las reglas son escritas en scripts utilizando un metalenguaje de scripting utilizando un proceso llamado scripts de flujo de diálogo. Algunas de sus carácteristicas son: detección de significado utilizando patrones, diccionario de WordNet incorporado para ontología y corrección ortográfica, ontología con sustantivos, verbos, adjetivos y adverbios que puede ser extendida, las reglas pueden cambiar el comportamiento del motor y el script, recuerda las interacciones de los usuarios en las conversaciones, permite escanear documentos en busca de contenidos [36].

```
#
# file: food.top
#
topic: ~food []

#! I like spinach
s: ( I like spinach ) Are you a fan of the Popeye cartoons?

       a: ( ~yes )  I used to watch him as a child. Did you lust after Olive
Oyl?
           b: ( ~no ) Me neither. She was too skinny.
           b: ( yes ) You probably like skinny models.

       a: ( ~no ) What cartoons do you watch?
              b: ( none ) You lead a deprived life.
              b: ( Mickey Mouse ) The Disney icon.

#! I often eat chicken
u: ( ![ not never rarely ] I * ~ingest * ~meat ) You eat meat.

#! I really love chicken
u: ( !~negativeWords I * ~like * ~meat ) You like meat.

#! do you eat bacon?
?: ( do you eat _ [ ham eggs bacon] ) I eat '_0

#! do you like eggs or sushi?
?: ( do you like _* or _* ) I don't like '_0 so I guess that means I prefer
'_1.

#! I adore kiwi.
s: ( ~like ~fruit ![~animal _bear] )  Vegan, you too...

#! do you eat steak?
?: ( do you eat _~meat ) No, I hate _0.

#! I eat fish.
s: ( I eat _*1 > )
  $food = '_0
  I eat oysters.
```

Ejemplo de script de Chatscript [36]

2.6 Clasificación de chatbots

Los chatbots pueden clasificarse en varias categorías dependiendo de varios criterios, su diseño y el propósito por el cual han sido diseñados [30, 54]:

- Basado en recuperación de información / Basado en generación
- Conversaciones cortas / Conversaciones largas
- Dominio abierto / Dominio cerrado

Los chatbots basados en recuperación de información utilizan un conjunto de respuestas predefinidas o una base de conocimiento. Para elegir la respuesta se puede utilizar un algoritmo de selección. Los modelos de generación crean respuestas aplicando un conjunto de técnicas, generalmente se basan en técnicas de traducción automática (por ejemplo redes seq2seq). Estos modelos consiguen tener una cantidad de respuestas mayor y más variada incluso en los casos en los que el chatbot no conoce la respuesta, pero son difíciles de entrenar y requieren muchos datos para ello.

En las conversaciones cortas se produce una sola respuesta en respuesta a una sola entrada. En las conversaciones largas se intercambia mucha información que debe guardarse para obtener resultados en las siguientes interacciones con el usuario.

Las conversaciones de dominio abierto son conversaciones que cambian constantemente de dominio y no tienen ningún propósito en concreto. En las conversaciones de dominio cerrado el conocimiento del chatbot es limitado y no suele permitir desviarse del propósito específico para el que ha sido diseñado (por ejemplo, un sistema de atención al cliente).

Esta clasificación no es estricta, pueden desarrollarse chatbots que funcionen con recuperación de información y generación [46], conversaciones cortas y largas y con dominio cerrado pero permitiendo conversaciones cortas sin objetivo alguno (“Small talk”). Existen distintas arquitecturas de chatbot documentadas, pero todavía no ha surgido ningún modelo universal de cómo debe diseñarse un chatbot [55].

2.7 Herramientas y servicios actuales para creación de chatbots y asistentes con NLP

Existen bastantes bases de conocimiento (ontologías), bibliotecas y servicios (en local y en la nube) para realizar o ayudarnos a realizar procesamiento del lenguaje natural. A continuación se describirán las más populares y utilizadas actualmente.

2.7.1 Bases de conocimiento

WordNet. Es una gran base de datos léxica de inglés. Los sustantivos, verbos, adjetivos y adverbios se agrupan en conjuntos de sinónimos cognitivos (synsets), cada uno de los cuales expresa un concepto distinto. Los conjuntos están interrelacionados por medio de relaciones conceptuales-semánticas y léxicas. La estructura de WordNet hace que sea una herramienta útil para la lingüística computacional y el procesamiento del lenguaje natural. Es libre de descarga y modificación [57].

Cyc. Es una gran base de conocimientos que contiene palabras, reglas y conceptos que permite realizar razonamientos humanos. Un ejemplo de razonamiento: "Si llueve el suelo está mojado" [82].

2.7.2 Bibliotecas

NLTK (Natural Language Toolkit). Es una plataforma para crear programas en Python capaces de trabajar con datos de lenguaje natural. Proporciona interfaces fáciles de usar a más de 50 corpus y recursos léxicos como WordNet, junto con un conjunto de bibliotecas de procesamiento de texto para clasificación, tokenización, derivación, etiquetado, análisis y razonamiento semántico. También provee interfaces para usar otras bibliotecas de lenguaje natural como Standford NLP [56].

spaCy. (Similar a NTLK) Los creadores de spaCy la definen (traducido literalmente) como: "Procesamiento de lenguaje natural de fuerza industrial". Los motivos: sobresale en tareas de extracción de información a gran escala y es la biblioteca más rápida en procesar tareas de NLP. También es una de las mejores maneras para preparar texto para los algoritmos de aprendizaje profundo [58].

Scikit-learn. Proporciona una amplia colección de algoritmos de aprendizaje supervisados y no supervisados. Incluye algoritmos de clasificación, regresión, clustering, reducción de la dimensión y preprocesado [59].

TensorFlow. Es una plataforma para el aprendizaje automático que facilita la creación y el despliegue de modelos. Cuenta con un ecosistema flexible de herramientas, bibliotecas y recursos [60].

Rasa. (Antes dividida en Rasa NLU y Rasa Core) **Rasa NLU** es una herramienta de procesamiento de lenguaje natural para la clasificación de intenciones y la extracción de entidades en chatbots. **Rasa Core** es un motor de diálogo para construir asistentes inteligentes, en vez de usar condiciones utiliza aprendizaje automático para predecir qué hacer [61].

Snips NLU. (Similar a Rasa NLU) Es una biblioteca de comprensión del lenguaje natural que permite analizar oraciones escritas en lenguaje natural y extraer información estructurada (intenciones, entidades, slots) [62].

Todas las bibliotecas mencionadas son software libre.

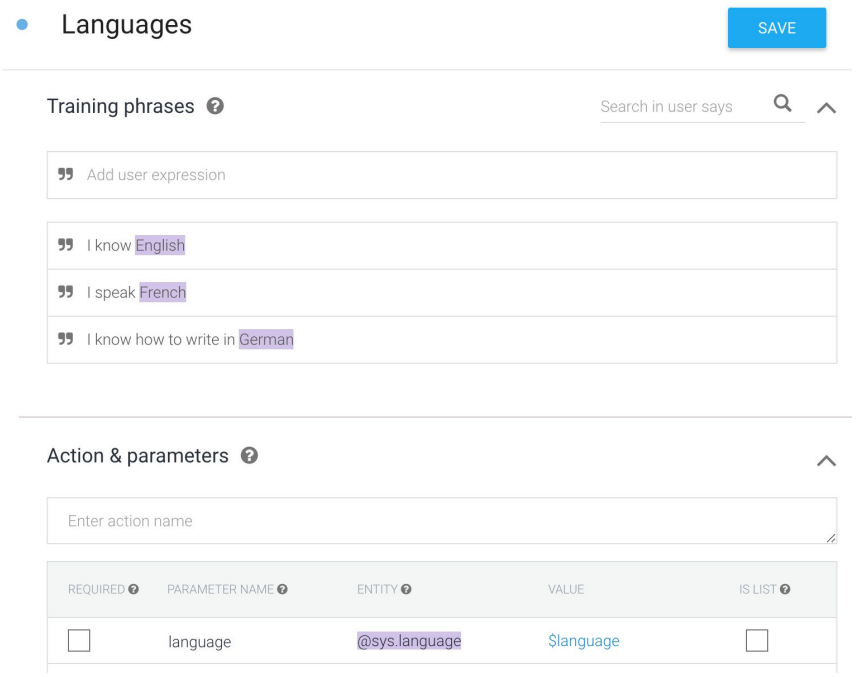
2.7.3 Servicios

Los principales servicios en la nube para construcción de chatbots que usan NLU son: **Dialogflow** (Google) (antes llamado API.ai), **IBM Watson**, **Amazon Lex, LUIS** (Microsoft), **Wit.ai** (Facebook) y **Rasa** (también puede utilizarse como servicio [61]). Todas ofrecen básicamente lo mismo, utilizando datos de ejemplo, el usuario puede entrenar un clasificador para obtener intents y entidades/slots. Soportan varios idiomas. Ninguno da información de qué tecnologías utiliza por debajo (excepto Rasa, que es software libre, utiliza spaCy, MITIE, Scikit-learn entre otras) [55, 2]. Google, Amazon e IBM también ofrecen otros servicios NLU en la nube para otros usos: Google Cloud Natural Language [68], IBM Watson Natural Language Understanding [69] y Amazon Comprehend [70].

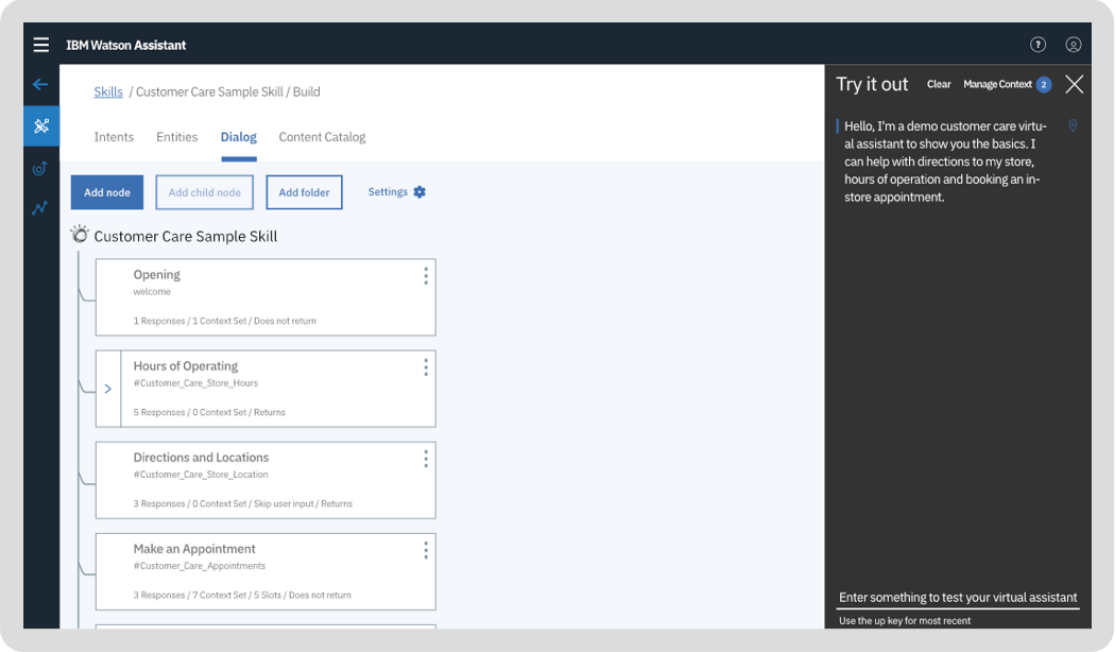
Dialogflow [65], IBM Watson [66] y Amazon LEX [64] ofrecen una interfaz gráfica para crear la conversación y conectar el chatbot a aplicaciones de mensajería populares y no solo crear los conjuntos de datos para la extracción de intents y entidades. Microsoft LUIS [67] y Wit.ai [68] no, solo ofrecen el análisis NLU.

El gestor de diálogo de Watson Assistant está formado por nodos, se utilizan condiciones para elegir el siguiente nodo al que ir en la conversación [66]. En Dialogflow y Amazon Lex un nodo se corresponde con un intent y se puede acceder a cada nodo desde cualquier otro nodo.

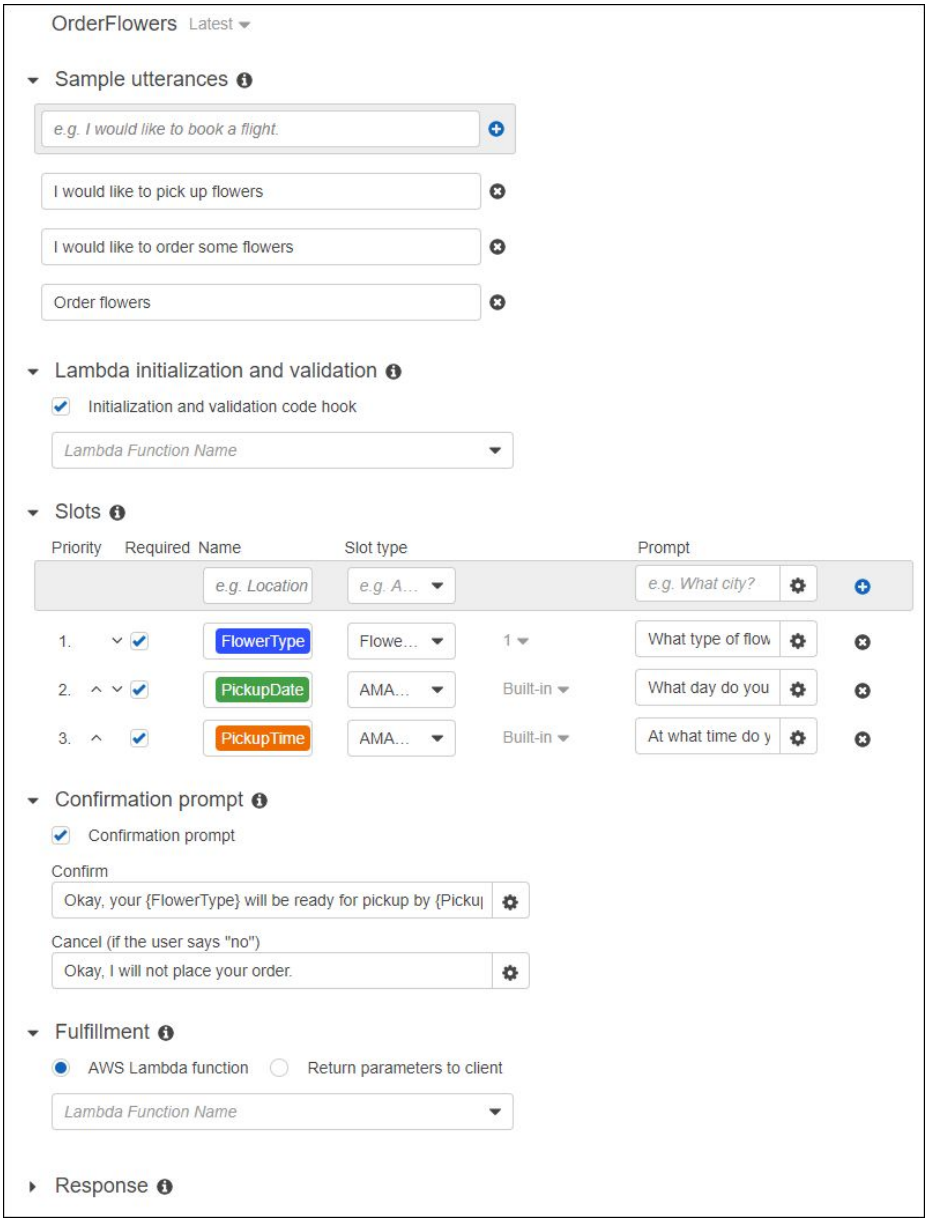
A continuación comparto unas capturas de pantalla obtenidas de la documentación de Dialogflow, Watson y Amazon para mostrar el aspecto de estas aplicaciones y poder hacerse una idea de cómo son.



Ejemplo de pantalla de edición de intents y entidades de Dialogflow [65]



Ejemplo de página del gestor de diálogo de IBM Watson Assistant [66]

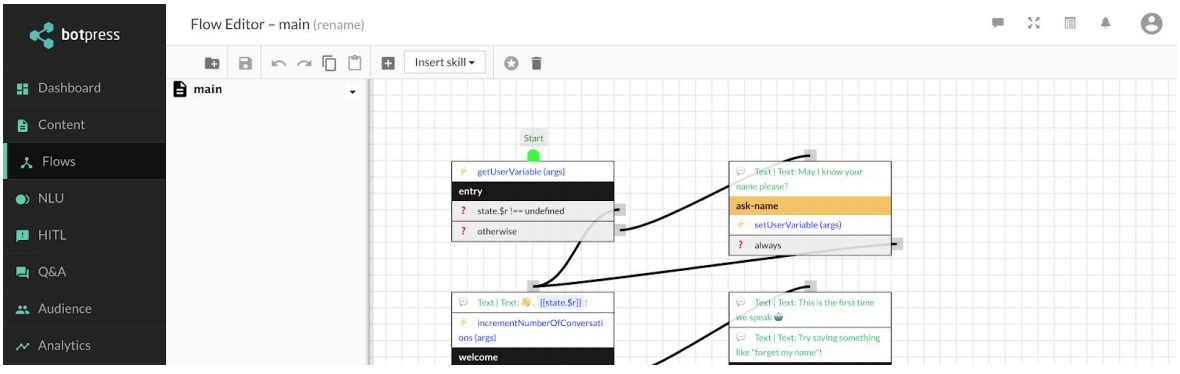


Ejemplo de página de Amazon Lex para edición de intents, slots y respuestas [64]

En la siguiente gráfica se compara el rendimiento de los servicios en la nube mencionados junto a Rasa y Snips. Cuanto más alto es el valor, mejor es el reconocimiento del lenguaje natural. Los resultados son de hace más de un año (entre agosto de 2017 y enero de 2018), por lo que actualmente estos datos han podido variar.

Comparación del rendimiento de varias soluciones de NLU [63]

Otra aplicación bastante completa para construir chatbots es **Botpress**. Tiene una versión libre que cuenta con un motor NLU, un editor de flujo visual, un emulador de chat y una página de administración [72].



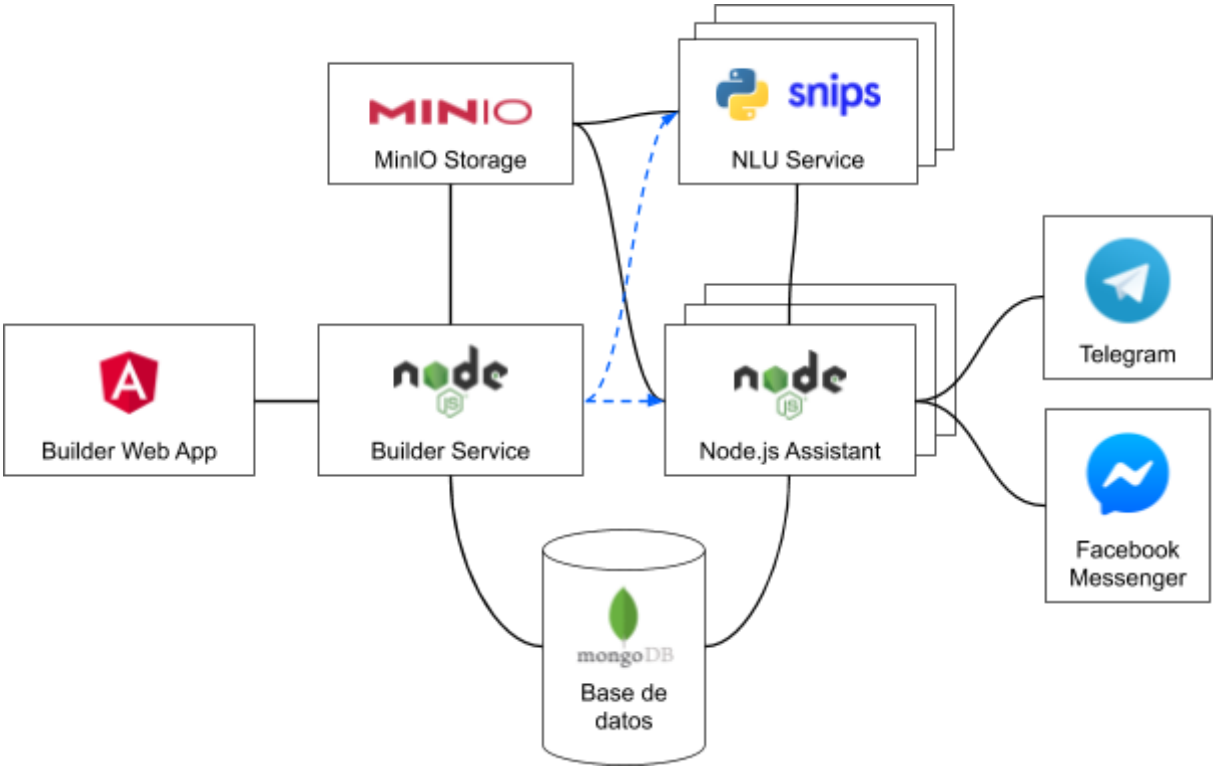
Ejemplo de página de editor de flujo de Botpress [72]

Existen otras muchas plataformas de creación de chatbots, como Pandorabots, Chatfuel, ManyChat, FlowXO, Botsify, por nombrar algunas. Una parte de estas plataformas o servicios no utiliza NLU (esto tampoco es malo, no siempre es necesario) y están enfocadas a estrategias de marketing dentro de Facebook [73, 74].

3. DISEÑO DE LA APLICACIÓN

3.1 Visión general

La aplicación global (a la que he llamado “Maiara”, nombre originario de los pueblos tupí-guaraníes de América del Sur y que significa “Una mujer de gran sabiduría”) es en realidad un conjunto de seis aplicaciones con las que podremos diseñar, entrenar e iniciar los chatbots para comunicarse con usuarios de las plataformas de mensajería Telegram y Facebook Messenger. Todas las aplicaciones funcionan dentro de contenedores Docker:



Arquitectura general de la aplicación

- **Builder Web App**. Aplicación web para diseñar los chatbots desde un navegador. Se comunica con el servicio "Builder Service" a través de una API REST.
- **Builder Service**. Servicio para obtener y guardar todos los datos de los chatbots (datasets para NLU, diálogo, configuraciones). También puede iniciar las aplicaciones "Node.js Assistant" y "NLU Service".
- **Node.js Assistant**. Servicio que recibe los mensajes de Telegram y Facebook Messenger y que utiliza uno o varios "NLU Service" y el diálogo diseñado con la aplicación web para devolver una respuesta al usuario.
- **NLU Service**. Usa los datasets creados en la aplicación visual para analizar y detectar los intents y entidades/slots en los mensajes de los usuarios. Utiliza la biblioteca "Snips NLU" para ello.
- **MinIO Storage**. MinIO es un servidor de almacenamiento de objetos compatible con el servicio de almacenamiento en la nube de Amazon S3. Los objetos pueden tener un tamaño de hasta 5 TB [75]. Es usado para guardar los datasets con los intents y las entidades, los diálogos y configuraciones.
- **Base de datos**. Se usa una base de datos NoSQL (MongoDB) para registrar qué aplicaciones ha iniciado el servicio "Builder Service" y los contextos de las conversaciones.

La versión de la aplicación presentada en este documento está pensada para funcionar en una sola máquina y con solo una instancia por servicio para simplificar el desarrollo de esta primera versión. Para el caso de "Node.js Assistant" es una instancia por chatbot/asistente y para "NLU Service" una instancia por Skill (defino más adelante que es una Skill). En futuras versiones se trabajará para que funcione en varias máquinas en la nube.

Para iniciar los servicios en local se usa Docker Compose. Para definir los servicios se utiliza el fichero docker-compose.yml cuya configuración puede verse en la siguiente página. En este fichero se definen 4 aplicaciones: "Builder Web App", "Builder Service", el servicio de almacenamiento con MinIO y la base de datos MongoDB.

```yaml
version: "3.7"
services:

   builder-webapp:
       build: ./builder-webapp
       image: maiara/builder-webapp
       ports:
           - "3080:80"
       restart: always
       depends_on:
           - builder-service

   builder-service:
       build:
           context: ./
           dockerfile: ./Dockerfile.builder-service
       image: maiara/builder-service
       network_mode: "host"
       environment:
           NODE_ENV: production
       restart: always
       depends_on:
           - storage
           - database

   storage:
       image: minio/minio:RELEASE.2019-08-21T19-40-07Z
       ports:
           - "9000:9000"
       volumes:
           - ~/.maiara/minio/data:/data
       command: server /data
       environment:
         MINIO_ACCESS_KEY: vidsH98hjAKJ
         MINIO_SECRET_KEY: jy0n0ASkjs623823jaJAKJSNJKD29239ihK
       restart: always
       healthcheck:
         test: ["CMD", "curl", "-f", "http://storage:9000/minio/health/live"]
         interval: 1m30s
         timeout: 20s
         retries: 3
         start_period: 3m

   database:
       image: mongo:4.2
       ports:
           - "27017:27017"
       restart: always
       environment:
           MONGO_INITDB_DATABASE: maiara
```

Configuración para iniciar servicios en local (docker-compose.yml)

Los servicios “Node.js Assistant” y “NLU Service” no se incluyen en este fichero porque los inicia el servicio “Builder Service” cuando recibe la petición desde la aplicación web. No obstante, es necesario crear las imágenes de Docker de estos dos servicios antes de que el servicio “Builder Service” intente crear los contenedores con dichas imágenes.

> Docker es una tecnología para crear contenedores. Los contenedores encapsulan el software y sus dependencias en una unidad estandarizada para el desarrollo del software que incluye todo lo que necesita para ejecutarse: código, bibliotecas, “runtimes” y compiladores, herramientas del sistema. Esto garantiza que la aplicación se ejecute siempre igual en cualquier entorno [76].
>
> Docker Compose es una herramienta para definir y ejecutar aplicaciones formadas por varias aplicaciones encapsuladas en contenedores Docker. Se utiliza un fichero en formato YAML para configurar los servicios [77].
>
> Kubernetes es una plataforma para automatizar la implementación, escalado y administración de aplicaciones en contenedores [78].

Antes de continuar definiendo el diseño de cada aplicación tengo que explicar varios conceptos necesarios para entender mejor su arquitectura.

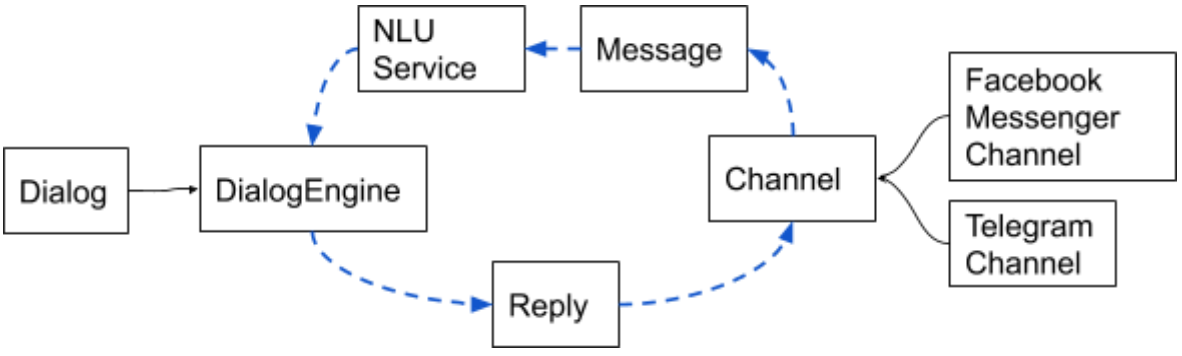
3.1.1 Asistentes y Skills

En la aplicación defino los chatbots/asistentes como Assistants. Los asistentes a su vez utilizan una o varias Skills (“Habilidades” en español). En otros asistentes como Google Assistant y Amazon Alexa los skills son aplicaciones de terceros que se integran con el asistente principal pero son independientes de él (excepto las Skills nativas del asistente). Usan la misma interfaz de comunicación (el móvil o el altavoz inteligente en el que se encuentra el asistente) pero el diálogo pertenece a otra aplicación. En mi aplicación los skills siempre pertenecen al asistente definido y la conversación no puede ser llevada por otros programas. Los skills pueden compartirse entre todos los asistentes que se quiera.

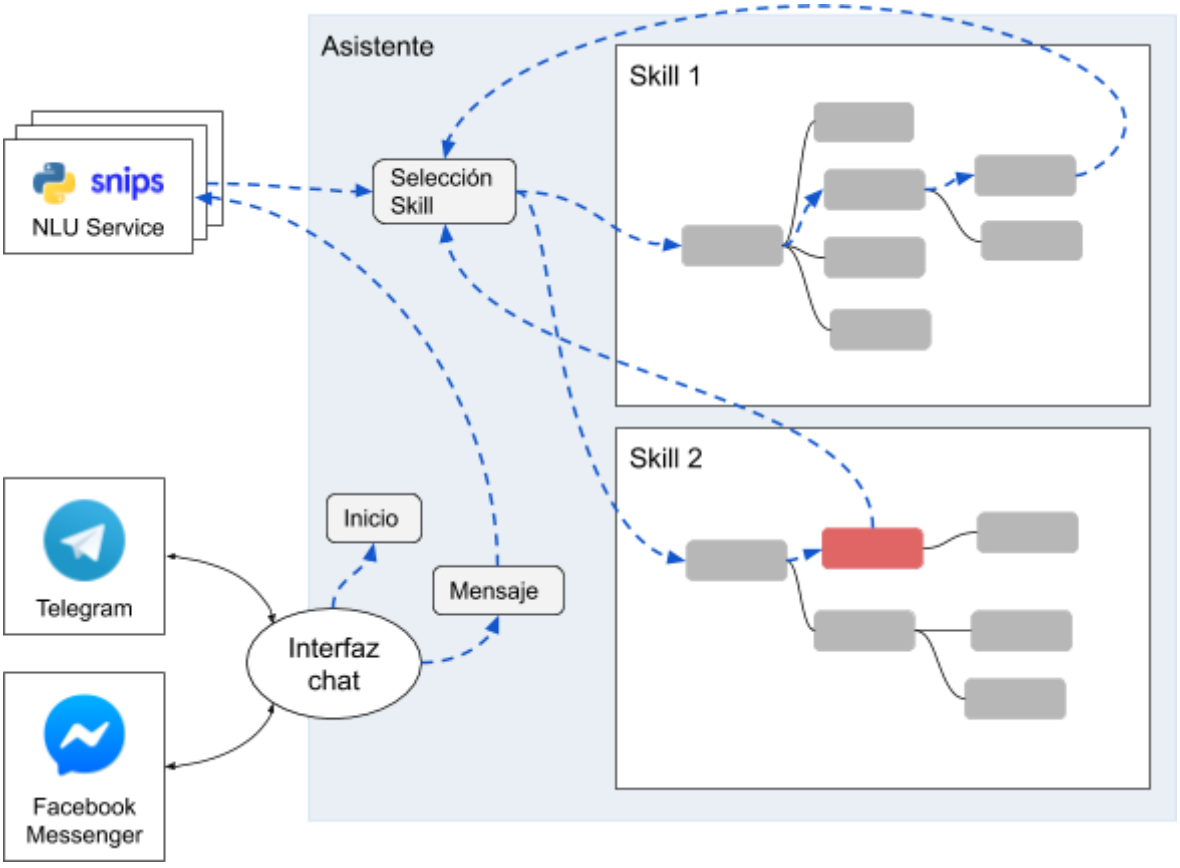
Un Skill está formado por uno o varios datasets (un dataset por idioma) y por flujos de diálogo (un flujo por idioma). Un dataset está formado por un conjunto de intents y entidades. Un flujo de diálogo está formado por nodos, en cada nodo se pueden realizar distintas acciones, como enviar una respuesta al usuario. Cada nodo también puede tener definidas varias condiciones para cambiar de nodo. Los datasets y los flujos de diálogo tienen exactamente la misma estructura en todos los idiomas, sólo varía el contenido (frases de ejemplo de los intents, valores de las entidades y respuestas del diálogo). Por ejemplo, queremos crear un Skill para español e inglés que reconozca preguntas como: "¿Qué tiempo hará mañana?". Cuando se crea un intent el usuario debe asignarle un nombre único, vamos a llamarle "GetWeather", este intent existe en todos los datasets con ese mismo identificador. Sin embargo, en el dataset en inglés tendremos frases de ejemplo como: "Tell me what the weather is going to be on Tuesday", "Is it going to be sunny tomorrow?", etc. y en español: "Dime qué tiempo va a hacer el martes", "¿Va a hacer sol mañana?", etc. No hace falta que las frases de ejemplo de cada idioma sean exactamente la traducción de un idioma a otro, solo que todas las frases tengan el mismo significado o intención. También puede variar el número de frases de ejemplo en cada dataset.

3.1.2 Gestión del diálogo por parte del asistente

El diálogo creado en la aplicación web es utilizado por un motor de diálogo. Cuando se recibe un mensaje de Telegram o Facebook Messenger, que implementan un interfaz de comunicación común, el mensaje se envía a un servicio de NLU, el resultado de este servicio es utilizado por el motor de diálogo para obtener una respuesta que es enviada al canal que envió el mensaje. El flujo se podría resumir con el siguiente gráfico:



Algunos chats envían un mensaje especial al iniciar una conversación por primera vez. Por ejemplo Telegram envía un mensaje con el texto "/start". Con Telegram se pueden utilizar comandos para solicitar a un bot que realice alguna acción. Los comandos siempre tienen el formato: "/textosinespacios". Cuando se recibe este mensaje de inicio nuestro asistente puede enviar un mensaje de bienvenida, algunos bots lo utilizan para contar al usuario que pueden hacer.

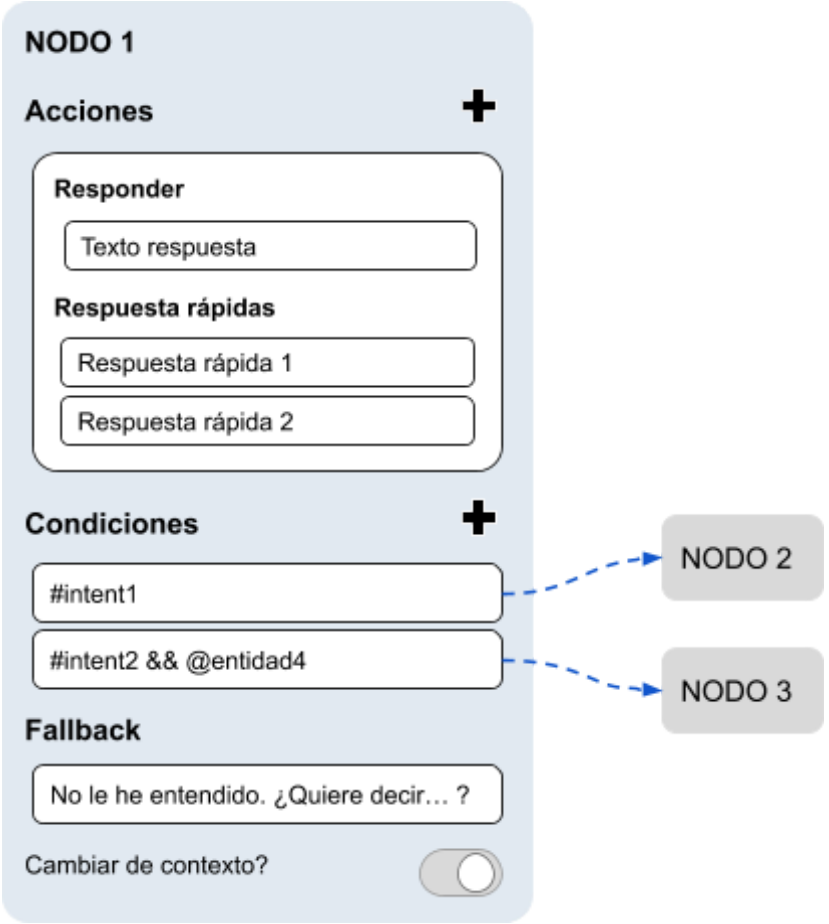


Flujo del diálogo

Cuando el usuario envía el primer mensaje, el asistente se encarga de elegir un Skill. Para ello envía el texto del mensaje a todos los servicios NLU de los Skill. El servicio NLU devuelve una probabilidad con cada intent detectado para indicar cómo de probable es que el usuario quiera decir lo que el servicio a detectado. El asistente elige la Skill cuyo servicio NLU devuelve la probabilidad más alta. Una vez que se ha elegido una Skill, es la Skill la que controla la conversación. Todos los Skills comienzan la conversación en un nodo "Start".

3.1.3 Funciones de un nodo del diálogo

Cuando la conversación entra a un nuevo nodo, el nodo ejecuta todas sus acciones. Después de ejecutar las acciones se espera a una respuesta del usuario. La respuesta del usuario es analizada por el servicio NLU y los resultados del análisis se usan en las condiciones. En las condiciones se pueden usar intenciones (utilizando "#nombreIntencion"), entidades (utilizando "@nombreEntidad"), slots (con @nombreEntidad[nombreSlot]), variables de contexto (con "$nombreVariable"), paréntesis, operadores comunes de cualquier lenguaje de programación (+, -, ==, &&, ||, etc.), cadenas de texto entre comillas, números, booleanos (true/false) y "null". Básicamente se puede usar cualquier expresión válida de JavaScript, excepto utilizar funciones.



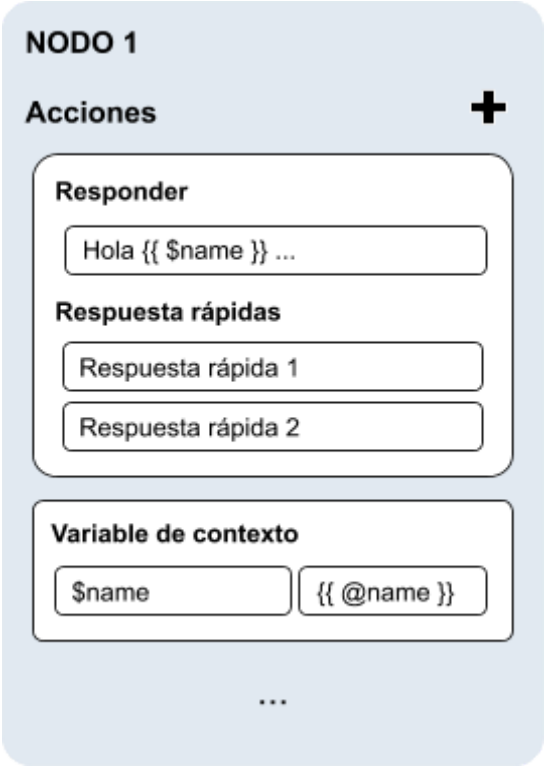
Ejemplo de estructura de un nodo

En la primera condición de la imagen de ejemplo, “#intent1” significa: “Si se detecta el intent1” y la condición 2, “#intent2 && @entidad4”: “Si se detecta el intent2 y la entidad4”. Las entidades y las entidades con slots se pueden comparar también con otros valores, por ejemplo “@datetime[weatherDate] == ‘today’ ”.

Cuando no se cumple ninguna de las condiciones del nodo, se puede usar una respuesta de “fallback” para comunicar al usuario que no se le ha entendido y que vuelva a escribir su mensaje, pudiendole sugerir cosas que puede decir.

Otra alternativa es cambiar de contexto. Con esto me refiero a que el asistente vuelva a elegir un nuevo skill según el resultado del servicio NLU. Si el asistente cambia de Skill, una vez que se ha finalizado el diálogo de esta, se puede volver al nodo en el cual se cambió el contexto si se desea. Solo se guarda un cambio de contexto, es decir, si desde la Skill seleccionada en el cambio de contexto, se vuelve a cambiar de contexto, la conversación no puede volver al nodo en el que se realizó el primer cambio.

Las variables del contexto se pueden utilizar para guardar información. Las variables son globales y pueden ser accedidas desde cualquier nodo de cualquier Skill. Se puede usar por ejemplo para guardar el nombre del usuario y utilizarlo en las respuestas. Para asignar un valor a una variables se utiliza una acción. En la imagen de abajo se asigna a la variable “name” el valor de la entidad con el mismo nombre.

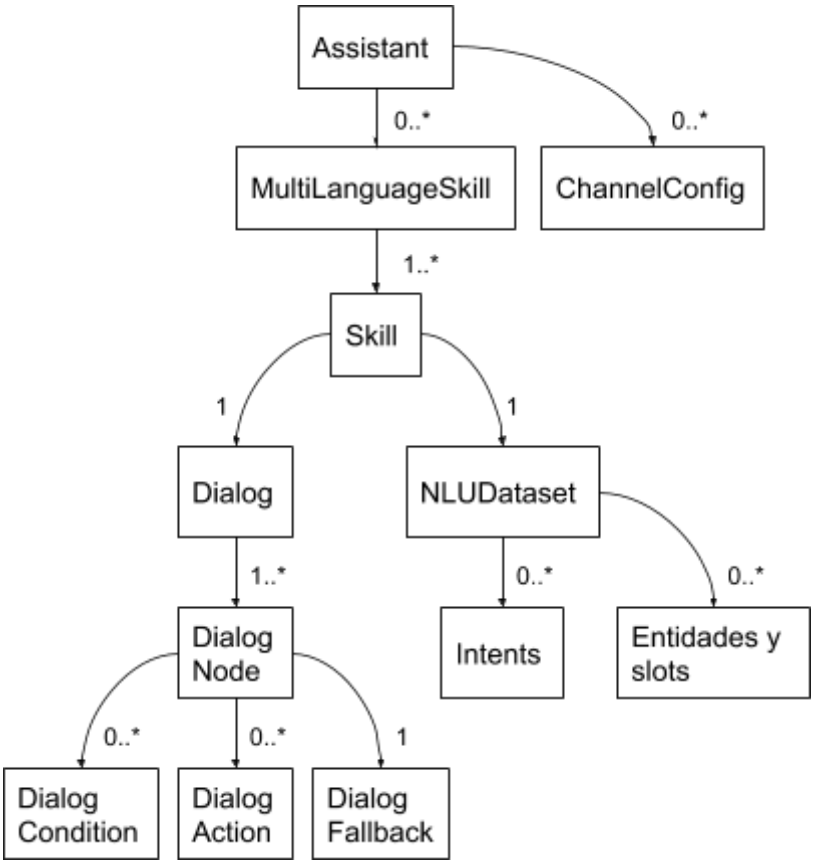


Ejemplo de uso de las variables de contexto

3.2 Paquetes comunes

Para mejorar la modularidad de las aplicaciones y facilitar la creación de nuevas funcionalidades, las aplicaciones desarrolladas con Node.js tienen algunas funcionalidades separadas en paquetes (módulos) independientes de las aplicaciones.

El paquete principal es el paquete “core”. Este paquete no depende de ningún otro paquete de la aplicación. En el están definidos todas las interfaces básicas y alguna funcionalidad común o de ayuda. Las interfaces más destacables pueden verse en el gráfico de más abajo. La interfaz Assistant define que skills va a utilizar el asistente, de qué chats va a recibir mensajes, un mensaje de bienvenida y una respuesta alternativa en caso de que los skills no consigan dar una. ChannelConfig contiene configuraciones para conectarse a los chats, como por ejemplo los tokens de seguridad. MultiLanguageSkill en una interfaz que encapsula a uno o varios skills, porque la interfaz Skill solo tiene definido un idioma. La interfaz Skill está compuesta por la interfaz Dialog y la interfaz NLUDataset. En NLUDataset guardamos todos los intents y las entidades y slots. La interfaz Dialog contiene todos los nodos de la conversación. Los nodos se definen con la interfaz DialogNode, que a su vez se compone de DialogCondition, para indicar en qué casos se puede pasar la conversación a otro nodo, DialogAction para definir acciones como enviar un mensaje y DialogFallback, en la que se pueden definir también mensajes alternativos en caso de que el servicio NLU no reconozca las intenciones o entidades que esperamos, o si queremos intentar cambiar de contexto (pasar la conversación a otro Skill).



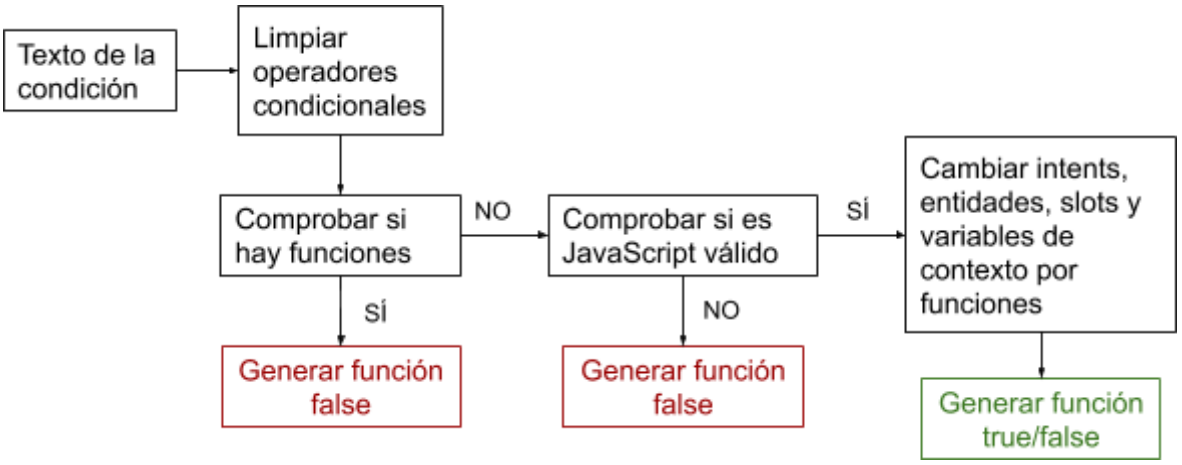
Algunas interfaces definidas en el paquete “core” y cómo se relacionan.

Otras interfaces definidas en este paquete son: Channel, para que todos los canales de comunicación / chats se comporten de las misma manera, y Storage para abstraer cómo obtienen y se guardan los datos.

Además del paquete “core”, están los paquetes “telegram” y “facebook-messenger” que implementan la interfaz “Channel” definida en el paquete “core”, y también el paquete “minio” que utiliza la interfaz “Storage” junto a la biblioteca de JavaScript de MinIO.

Algunas funcionalidades que se han implementado en el paquete “core” son el motor de diálogo (DialogEngine), con la funcionalidad explicada en la sección anterior y el parseador de condiciones del nodo (ConditionParser).

El parseador de condiciones convierte una condición en formato texto en una función que devuelve “true” o “false”. Funciona de la siguiente manera:



Primero se limpian los operadores condicionales (== && ||). Si solo hay un "=" se convierte en un "===" (en JavaScript el triple igual significa mismo valor y mismo tipo), si solo aparece un "&" se añade otro "&&", igual con "|", se convierte en "||". Si aparecen más veces repetidos (====, &&&, |||||, etc.) se borran los caracteres sobrantes. A continuación se comprueban si se ha escrito una función. Para comprobar si hay funciones se utiliza una expresión regular que comprueba si hay una palabra seguida de un paréntesis de apertura, puede haber espacios entre ambos. En el caso de que haya una posible función, se genera una función que devuelve siempre el valor "false". Después de limpiar caracteres y comprobar si hay funciones se utiliza la biblioteca "Esprima" para comprobar que el texto de la condición es código JavaScript válido. Esprima es un parseador de ECMAScript (JavaScript) escrito en JavaScript. Si el código no es válido se genera también una función que solo devuelve "false". Si es correcto se genera una función en donde los "#intentName", "@entityName", "@entityName[slotName]" y "$variableName" de la condición se reemplazan por unas funciones especiales que obtienen los valores del resultado del servicio NLU y del contexto. Esta función devuelve "true" o "false" dependiendo del resultado NLU y las variables guardadas en el contexto.

3.3 Almacenamiento

- Bucket
- Carpeta
- Fichero
- Comprimido ZIP

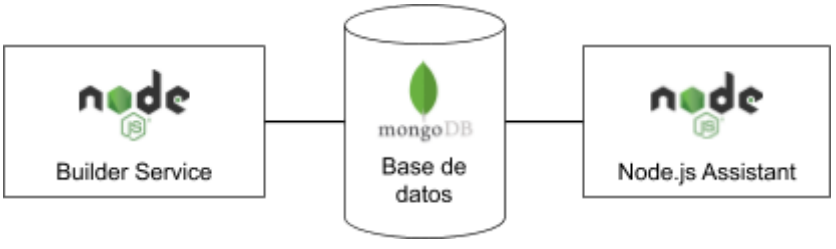
Estructura de ficheros para almacenar datasets, diálogos y configuraciones.

He utilizado MinIO como almacenamiento principal por ser un sistema sencillo de almacenamiento de objetos y por ser compatible con Amazon S3. De esta manera trasladar la aplicación a la nube será menos costoso. Se puede utilizar una base de datos NoSQL también, pero el límite máximo de tamaño del documento es mucho menor, por ejemplo, en MongoDB los documentos pueden ser de hasta 16 MB, en MinIO de hasta 5 TB. Si en algún momento se quieren crear datasets muy grandes, MinIO es una mejor opción. Todas las consultas que se hacen en la aplicación son de tipo clave-valor, por lo que no se van a percibir diferencias en el rendimiento.

En vez de tener todos los intents, entidades, nodos del diálogo y el resto de la información en un solo JSON, he decidido crear una estructura de directorios para que sea menos costoso modificar cada parte, además de que me parece más fácil encontrar los datos que necesito.

Los asistentes se encuentran en un directorio "assistants". Cada directorio de asistente tiene un "info.json" con los datos necesarios para que los servicios "Builder Service" y "Node.js Assistant" generen el asistente. Cada asistente puede tener varias configuraciones por entorno, en esta configuración solo se pueden definir por el momento la configuración para los chats.

Los skills se encuentra en un directorio con el mismo nombre. Cada skill puede tener distintas versiones. Dentro de cada versión hay otro directorio por idioma habilitado en el chatbot. Dentro del directorio del idioma se encuentra un directorio "dataset" con los intents y las entidades, y un fichero comprimido "train.zip" con el resultado del entrenamiento del servicio NLU. En el directorio "dialog" se guardan los nodos del diálogo. En el fichero "edges.json" se guardan las aristas que conectan los nodos, estas aristas solo se utilizan en la aplicación web para pintarlas en pantalla, el servicio "Node.js Assistant" solo necesita los nodos.



En la base de datos de MongoDB, "Node.js Assistant" guarda el contexto de la conversación. El contexto está compuesto por el identificador del Skill activo, el identificador del nodo activo, el idioma que se está usando y las variables de contexto. "Builder Service" guarda los identificadores de los contenedores Docker que ha iniciado, para luego poder detenerlos, y el identificador del asistente vinculado a esos contenedores.

3.4 Aplicación visual

La aplicación visual consiste en una aplicación web progresiva (PWA) desarrollada con Angular 8. Una aplicación web progresiva es una página web normal que se comporta igual o parecido a una aplicación de escritorio, pueden entre otras cosas, enviar notificaciones y funcionar sin conexión a Internet. Con algunos navegadores y sistemas operativos, las PWA pueden instalarse como una aplicación más (no es una instalación como una aplicación nativa, siempre se ejecutan en un navegador, pero la apariencia de la ventana es de una aplicación nativa y aparece el icono de la aplicación en los menús con el resto de aplicaciones) [79].

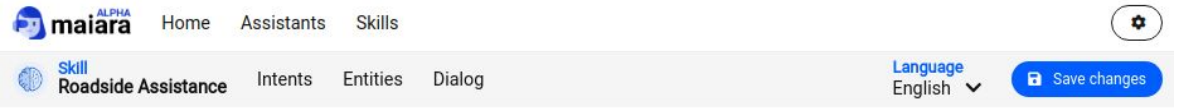
Angular tiene implementado un "service worker" que nos permite cachear fácilmente todos los ficheros que componen la aplicación para que las siguientes cargas sean casi instantáneas. Cuando se sube una nueva versión de la aplicación al servidor, este "service worker" se la descarga en segundo plano y muestra la nueva versión la siguiente vez que se recarga la página.

Un "service worker" es un script de JavaScript, que actúa como proxy del lado del cliente y permite controlar la caché y cómo responder ante las solicitudes de recursos [79].

La aplicación también tiene un diseño adaptable o como se suele llamar "responsive design". Este tipo de diseño consiste en utilizar HTML y CSS para redimensionar, reducir o ampliar automáticamente un sitio web, para que se vea bien en todos los dispositivos (escritorio, tablets y teléfonos) [80]. Algunos de los componentes creados en la versión presentada en este trabajo no se adaptan bien a pantallas de móvil, pero puede utilizarse en escritorio y tablets sin problemas. Todos los componentes funcionan con pantallas táctiles.

Solo barra global de navegación

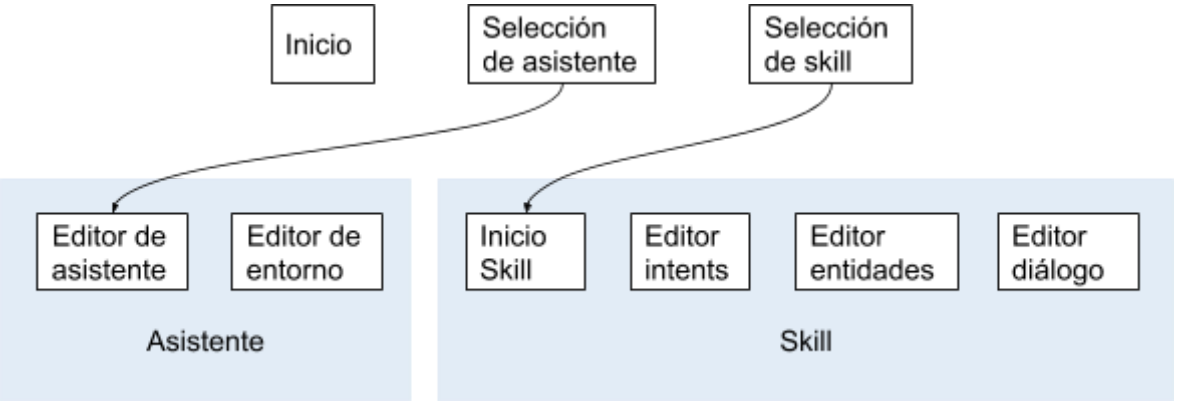
Barra global y barra de navegación de un asistente



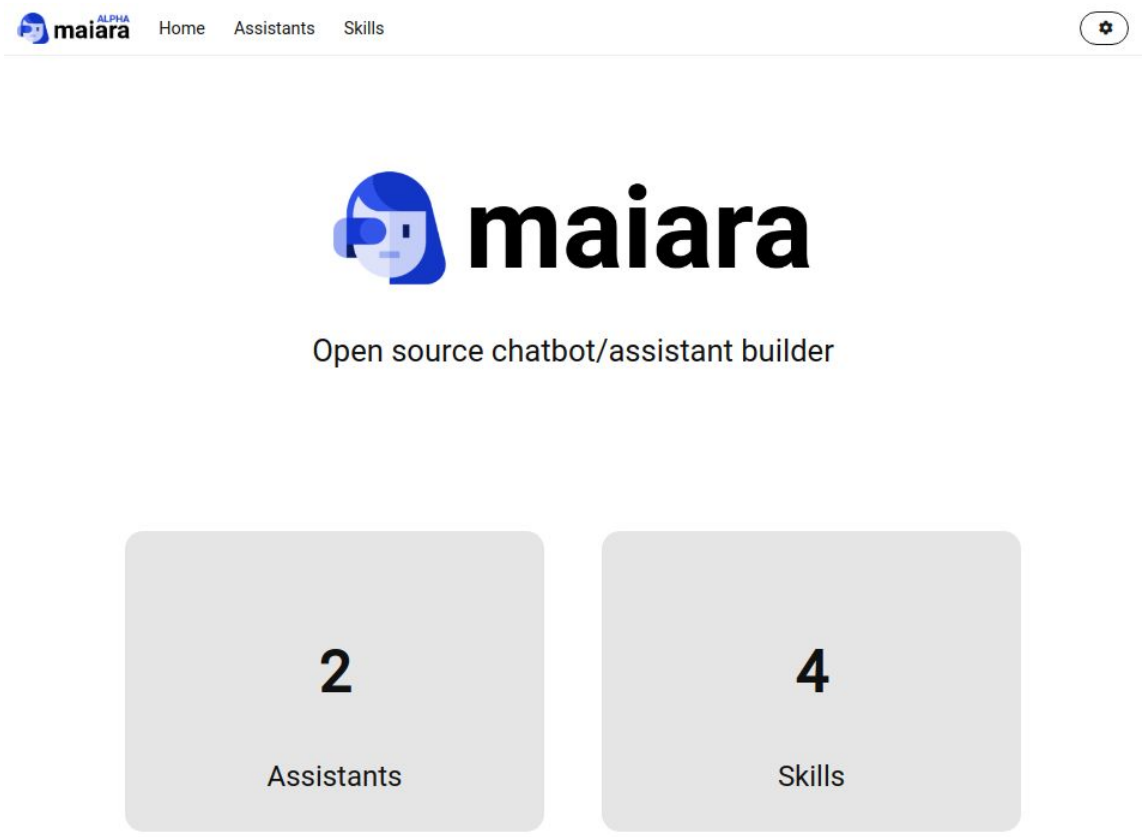
Barra global y barra de navegación de un skill

Para moverse por la aplicación existen tres menús superiores. El de más arriba es el menú global, con este menú se puede ir a la página principal, a la página de selección de asistente y a la página de selección de skill. Cada asistente y skill tiene otra barra de navegación. Con la barra del asistente podemos ir a la página principal del asistente y a una o varias páginas para editar variables de entorno. Con la barra de navegación de un skill podemos ir a la página principal de un skill, a la página de edición de intents, a la página de edición de entidades y a la página de edición del diálogo. Ambas barras de navegación tienen un botón para guardar los cambios realizados en el servidor. La barra del asistente tiene otro botón para iniciar el asistente utilizando un entorno y otro botón para crear más entornos. La barra del skill permite cambiar el idioma con el que se está trabajando.

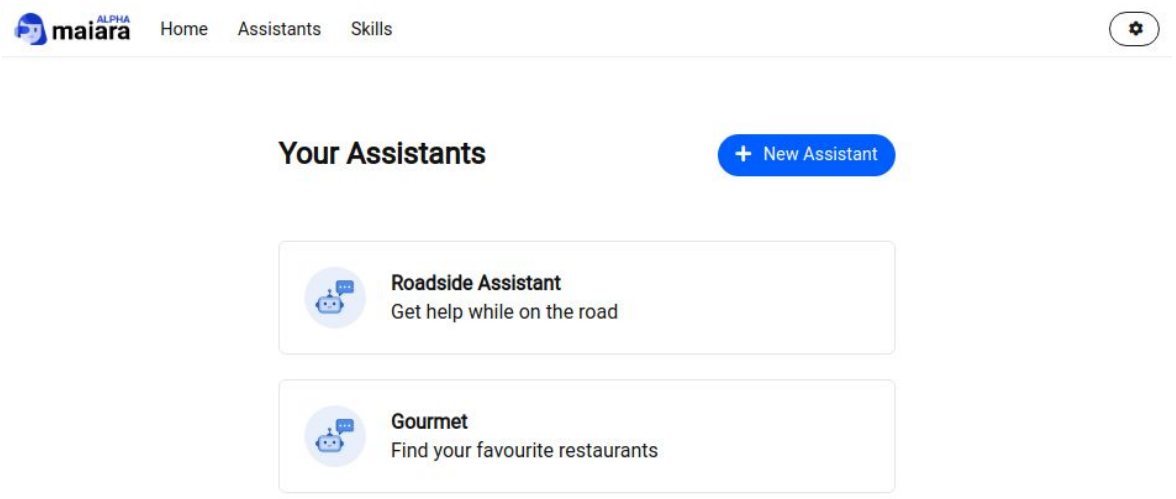
Las páginas de la aplicación pueden resumirse en el siguiente gráfico:



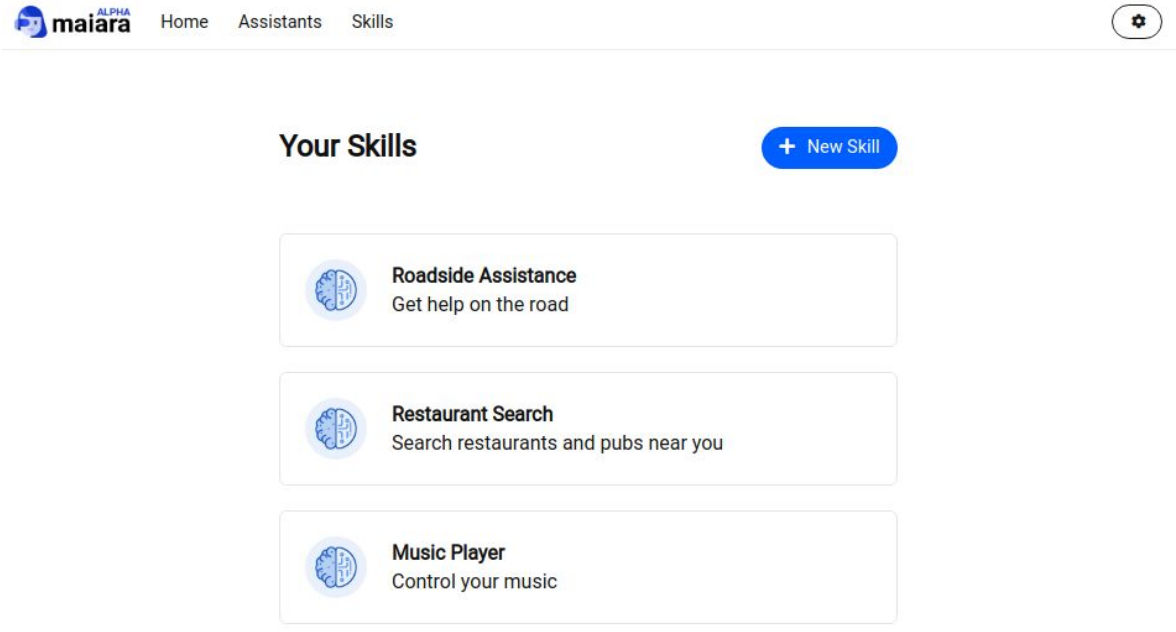
En las siguientes páginas del documento iré mostrando las páginas en más detalle.



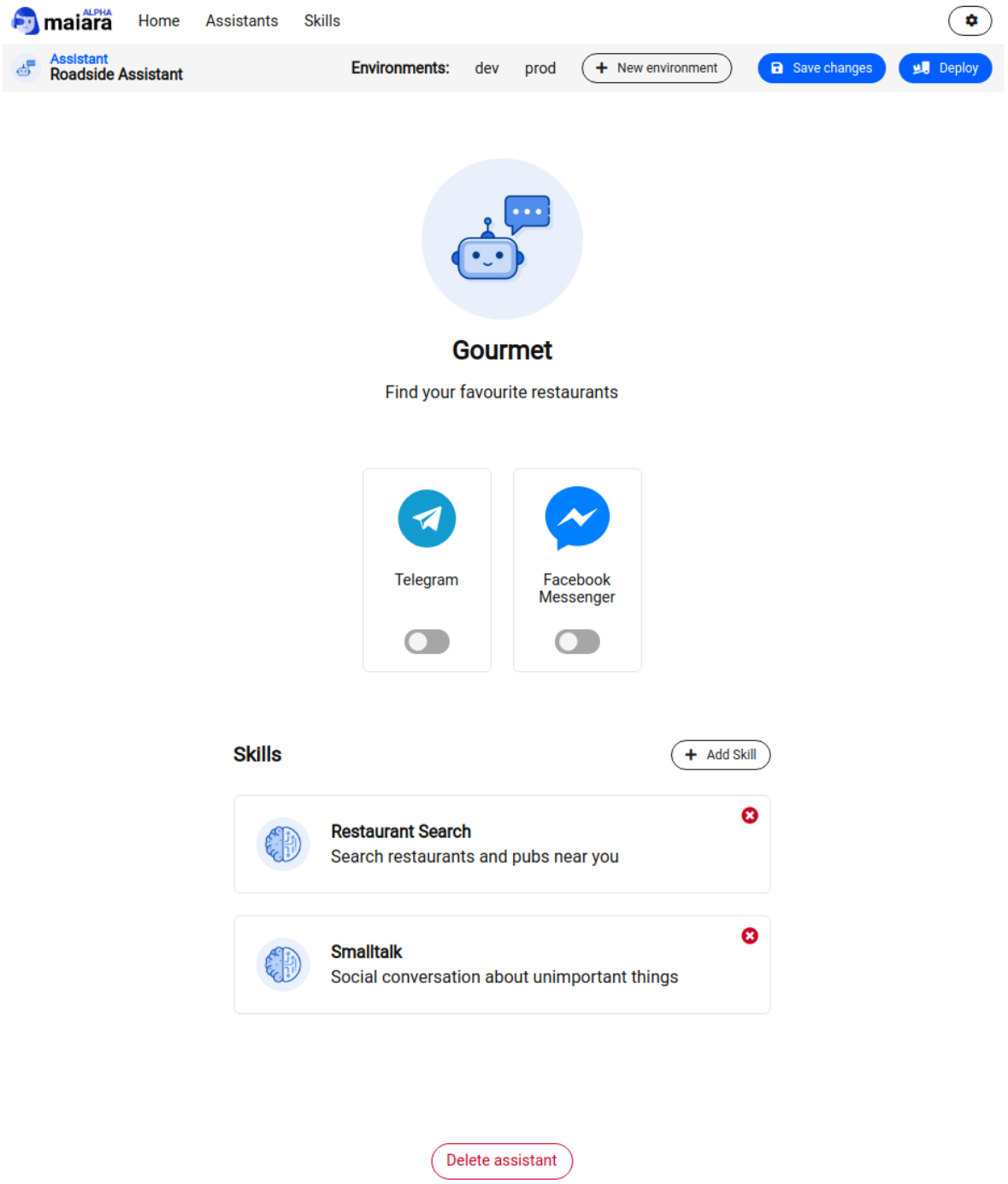
Inicio. Esta es la página inicial de la aplicación. Tiene el logo de la aplicación en grande y muestra el número de asistentes y skills creados. Al pulsar en el rectángulo con el número de asistentes se cambia a la página de selección de asistentes, y cuando se pulsa el rectángulo con el de skills se muestra la página de selección de skills.



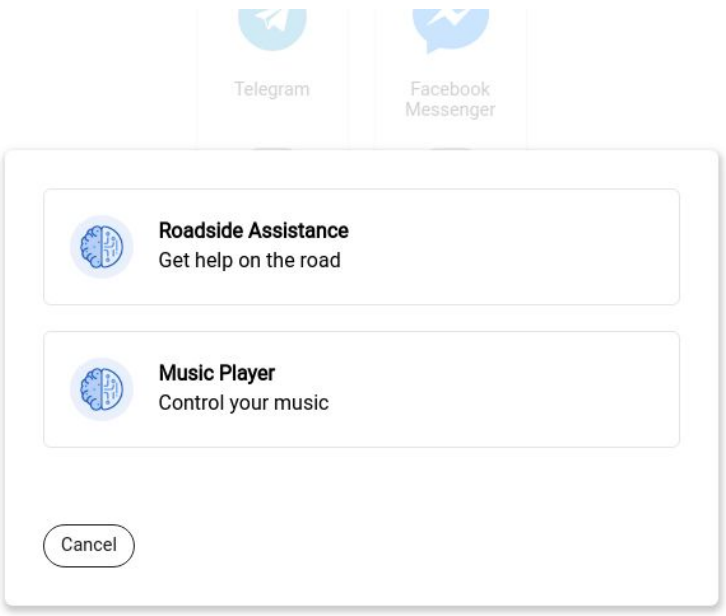
Selección de asistente. Se muestra una lista de todos los asistentes creados. Cada ítem de la lista tiene una imagen (genérica por ahora) con el nombre y la descripción del asistente. Podemos crear un nuevo asistente pulsando el botón azul. Al pulsar el botón se muestra un modal/popup para introducir el identificador, el nombre y la descripción del nuevo asistente.



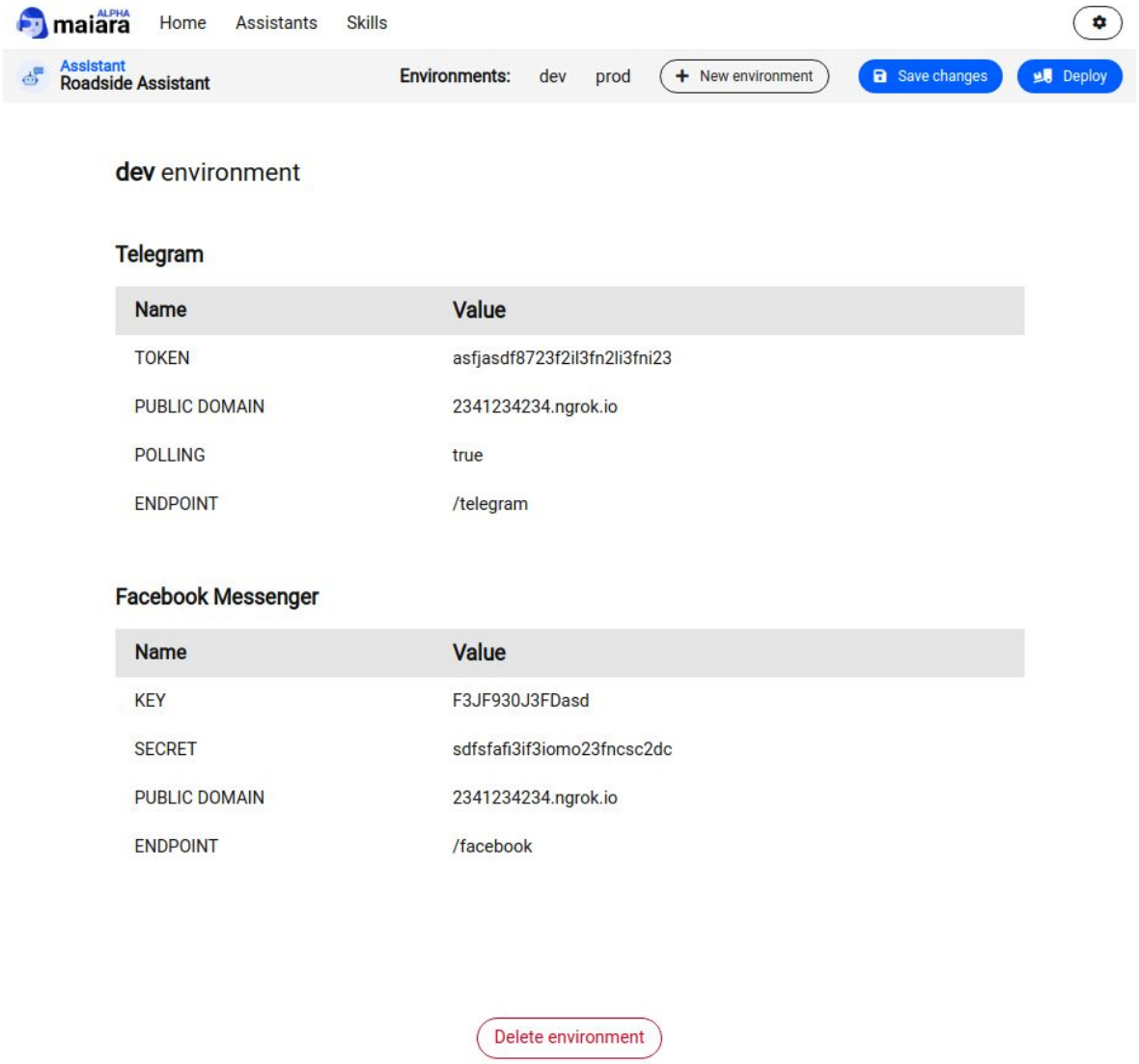
Selección de skill. Esta página es igual que la de selección de asistente pero con skills. Al pulsar el botón azul también se muestra un modal/popup para rellenar el identificador, el nombre y la descripción del skill que se quiere crear.



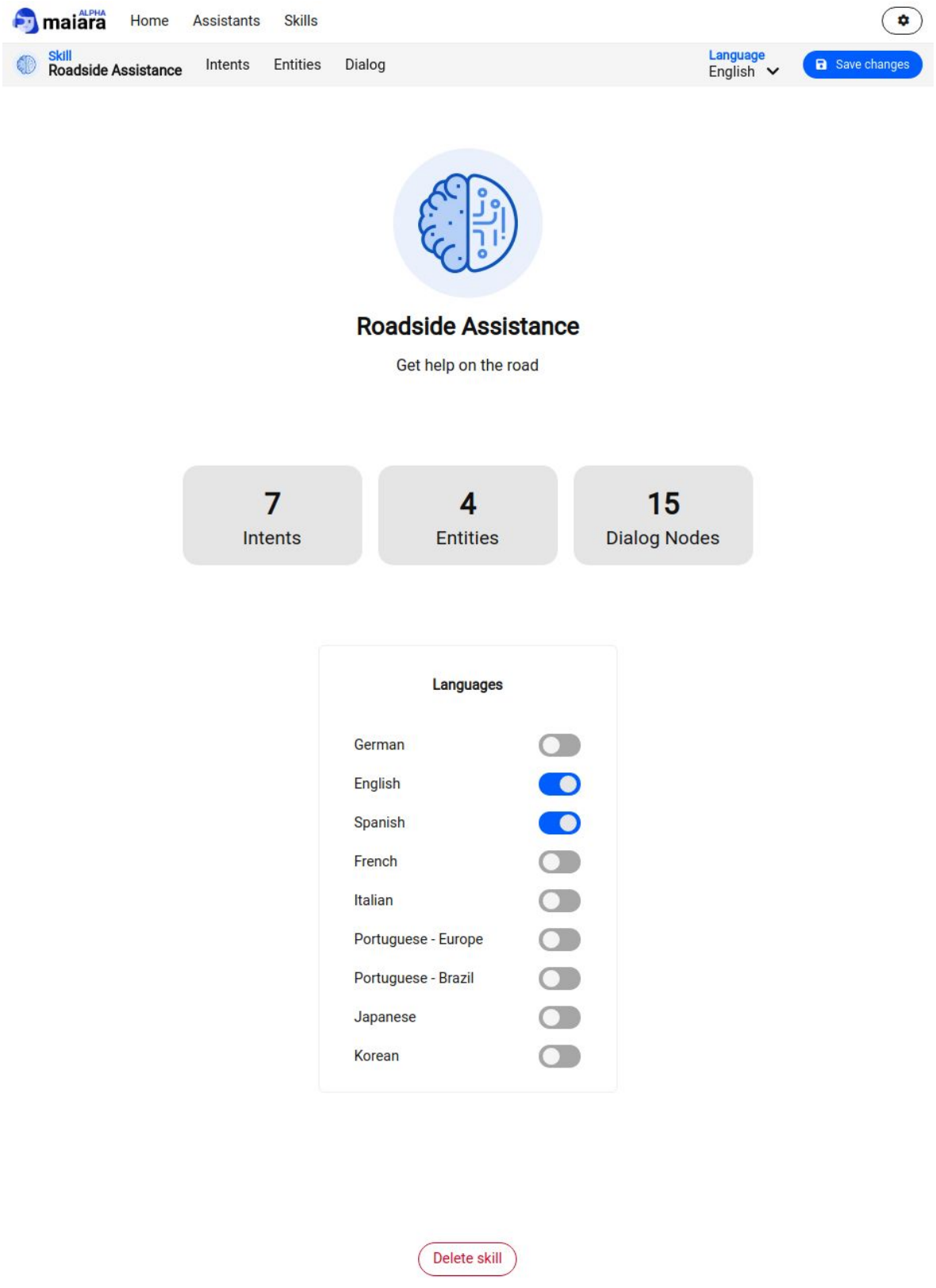
Página de inicio de un asistente. En esta página podemos editar el nombre y la descripción del asistente, podemos habilitar los chats de Telegram y Facebook Messenger y podemos añadirle o quitarle skills. También se puede eliminar el asistente con el botón rojo de abajo del todo.



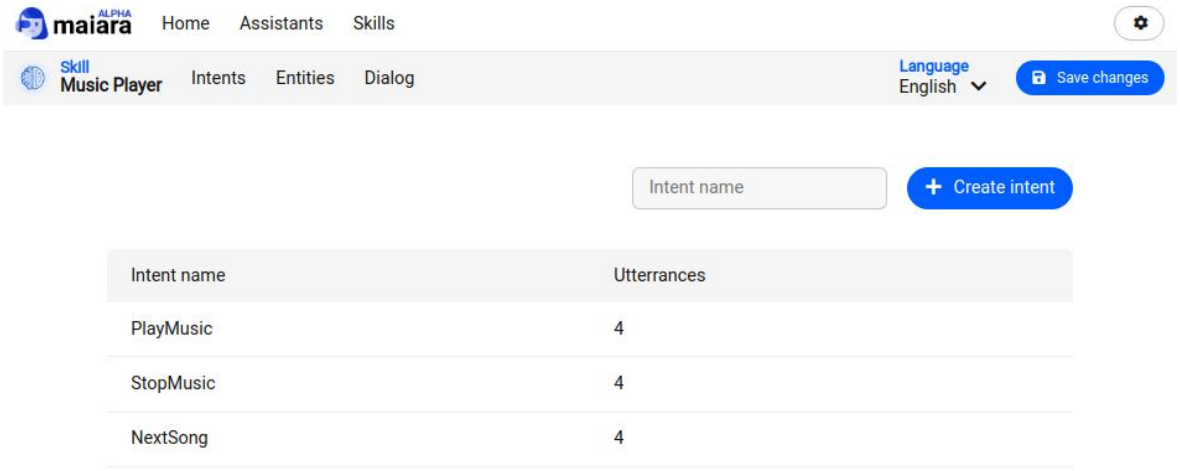
Cuando pulsamos el botón de añadir una skill se muestra un modal por encima de la página con una lista de todos los skills existentes y que no se han añadido ya.



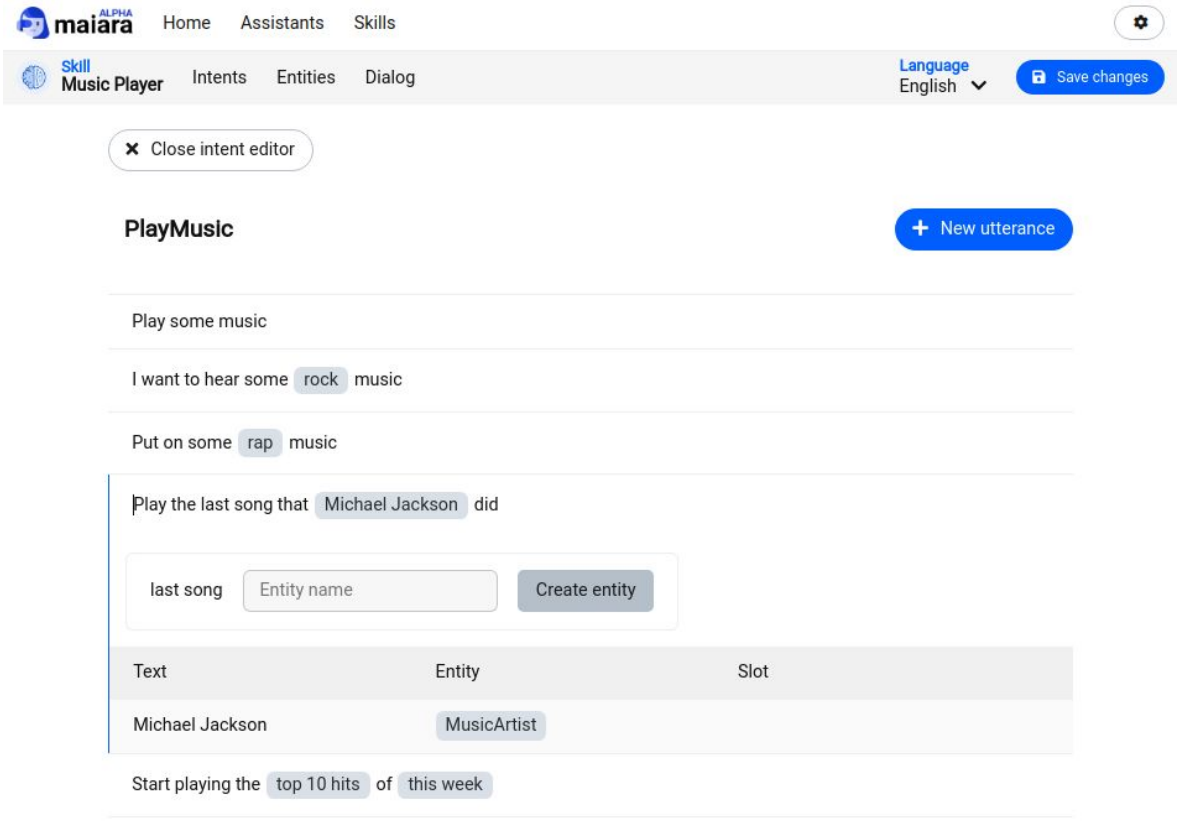
Página de edición de entorno. Si pulsamos en alguno de los entornos creados ("dev" y "prod" en este ejemplo) se muestra la página de edición de variables de entorno. Estas variables se usan para añadir la configuración necesaria para conectarse a servicios externos, en este caso de Telegram y Facebook Messenger. La razón por la que hay más de un entorno es que de esta manera puedes tener un asistente funcionando en un chat de pruebas y otro en un chat que atienda a clientes reales.



Página principal de una skill. Se puede editar el nombre y la descripción de la skill, habilitar o deshabilitar idiomas y borrar la skill. Los idiomas de la imagen son los idiomas soportados por Snips. También se muestra el número de intents, entidades y nodos de la skill.

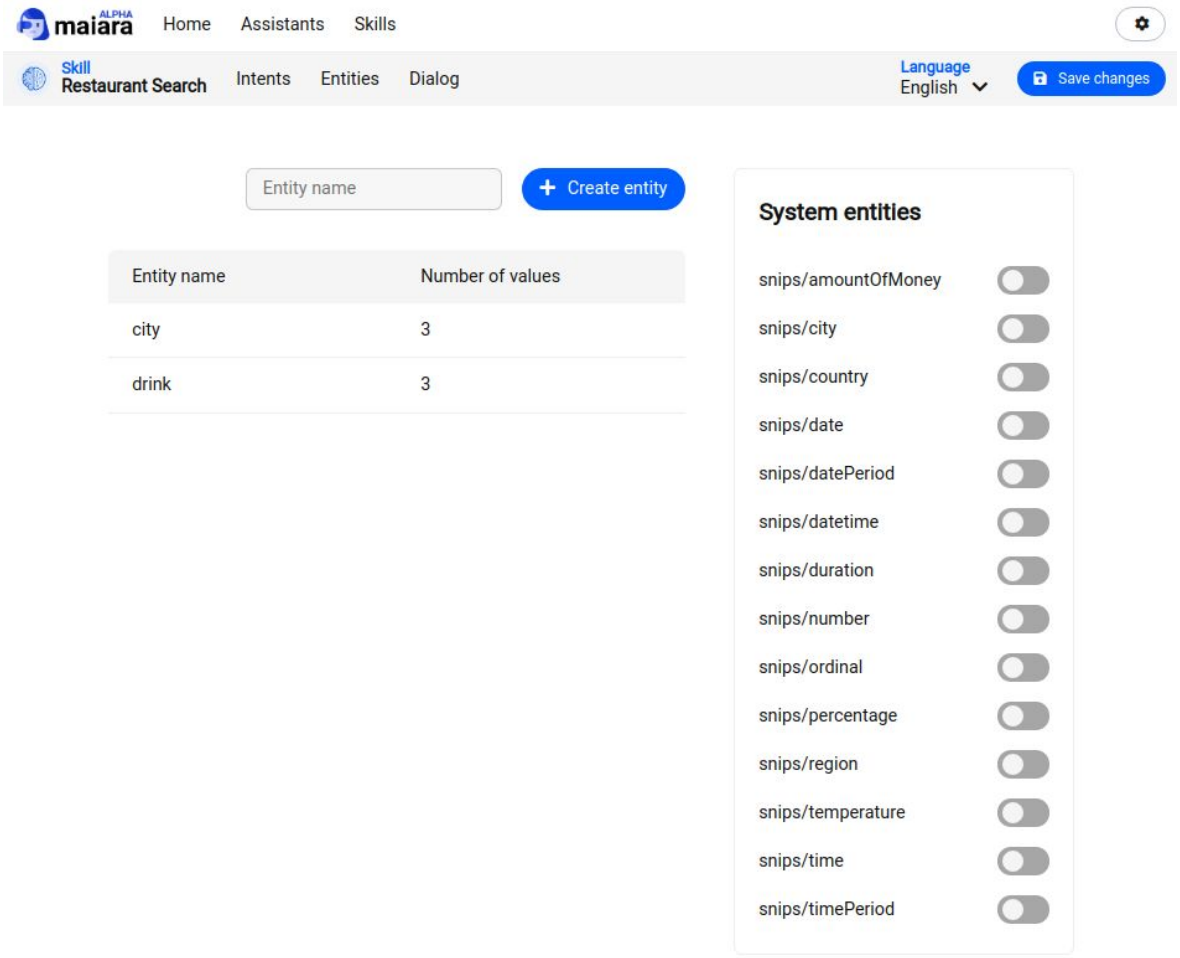


Página de edición de intents. En esta página se muestra una lista con los intents creados. Cada fila de la lista muestra el nombre del intent y el número total de frases de ejemplo (utterances) del intent. Para crear un nuevo intent escribimos su nombre en la entrada de texto de arriba a la derecha y pulsamos el botón azul para crearlo.

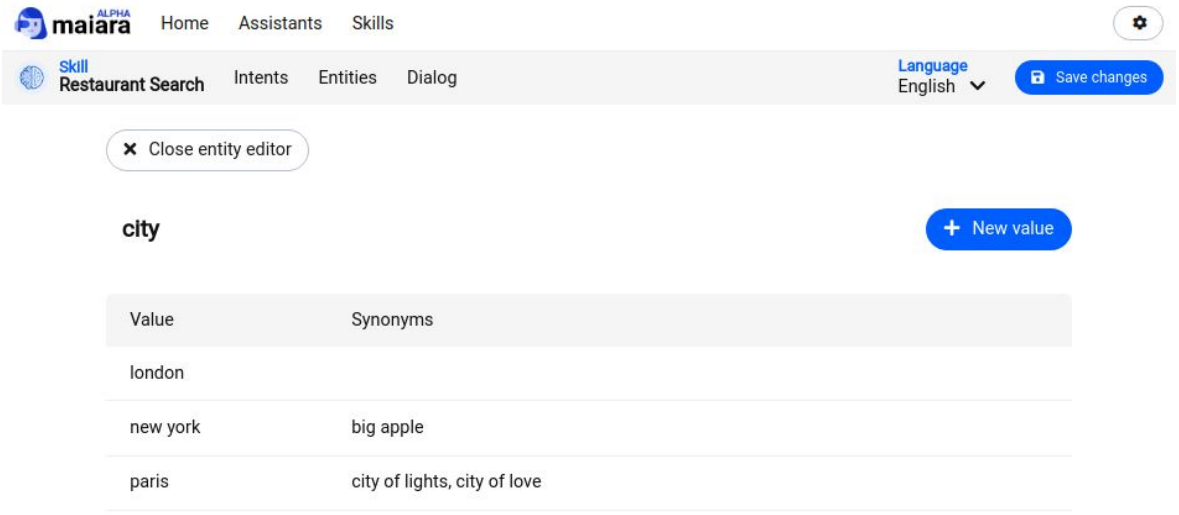


Si pulsamos en un intent de la lista de arriba, se abre el editor de frases de ejemplo. Cuando pulsamos en una frase se abre el editor de la frase. Con este editor podemos definir las entidades y slots que contiene la frase. Para definir una entidad se selecciona con el ratón las palabras que componen la entidad.

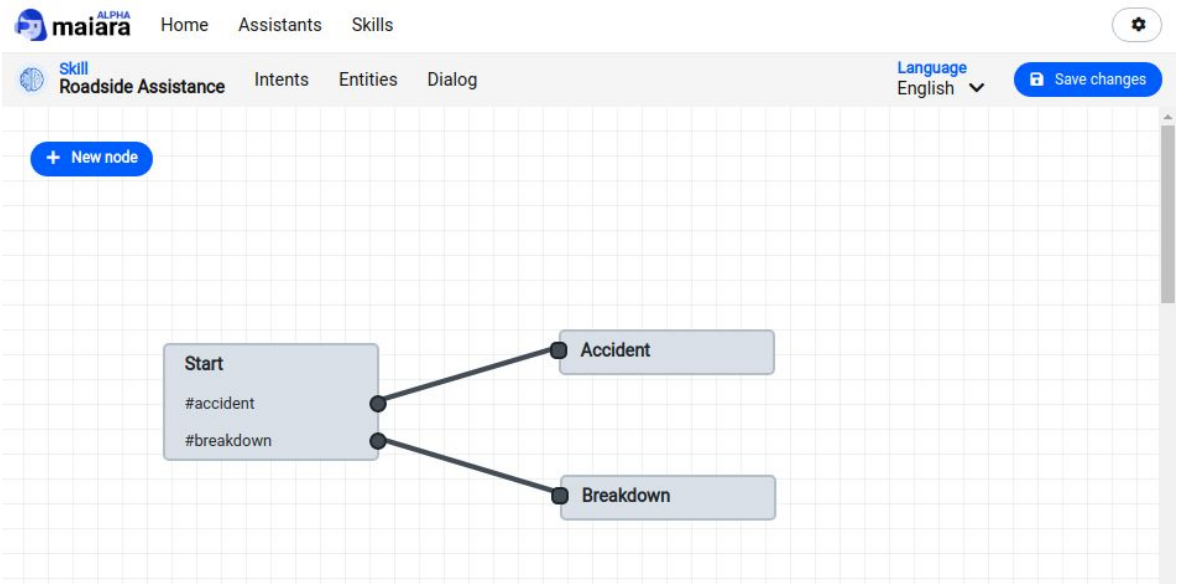
Aunque sólo se seleccionen unas letras se auto-selecciona la palabra entera. Cuando se ha seleccionado alguna palabra se muestra un formulario para introducir el nombre de la entidad. Cuando se crea la entidad se muestra en la tabla de abajo de la frase. En esta tabla podemos añadir también el nombre del slot si queremos. Cuando pasamos el ratón por encima de una frase o de un elemento de la tabla de intents, se muestra un icono con una cruz roja para poder borrar la frase o el intent.



Página de edición de entidades. En esta página podemos ver una lista con las entidades creadas. También podemos crear nuevas con el formulario que se encuentra encima de la tabla de entidades. A la derecha hay un panel con todas las entidades del sistema. Las entidades del sistema son entidades especiales de las que no tenemos que preocuparnos de crear, y sirven para detectar números, fechas, duración, temperaturas, etc. Las entidades de sistema que se muestran en la imagen son las entidades de sistema que Snips NLU es capaz de detectar.

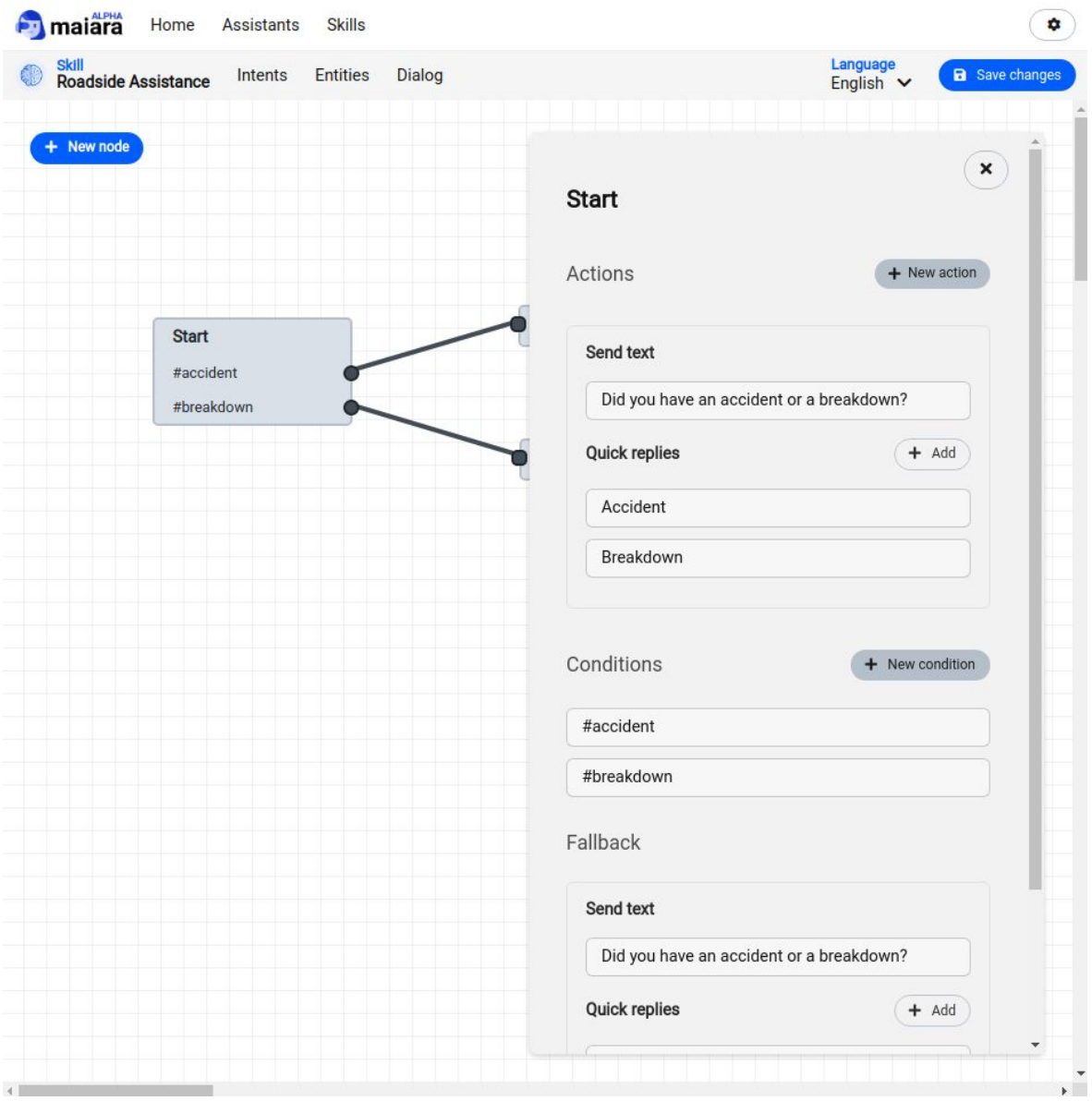


Cuando pulsamos en una entidad de las que hemos creado, se abre el editor de la entidad. En este editor podemos definir todos los valores posibles de la entidad y sinónimos de cada valor. Al igual que con los intents, cuando se pasa el ratón por encima de una fila de la tabla, se muestra un icono para poder borrar el valor.

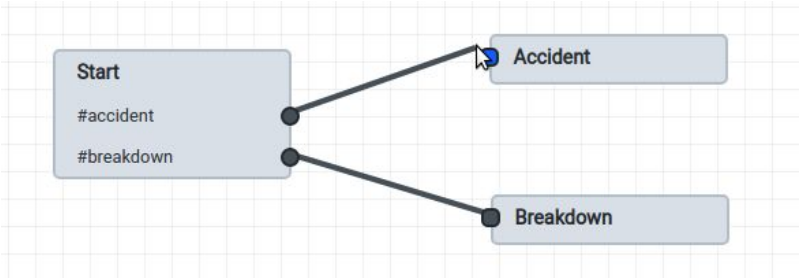


Página de edición de diálogo. En esta página podemos crear los nodos y las conexiones entre los nodos que dirigen la conversación. Todos los nodos tienen una entrada y ninguna o varias salidas. El nodo "Start" solo puede tener salidas. Es el nodo de entrada a la skill y no se puede borrar. Se crea automáticamente al crear la skill. Con el botón de arriba a la izquierda podemos crear nuevos nodos. Los nodos que se crean están vacíos de información. Para editar un

nodo debemos pulsar en uno de ellos para abrir el editor del nodo.



El panel flotante que aparece en la derecha es el editor del nodo. La lógica de este componente está explicada por el final de la sección “Visión general”.

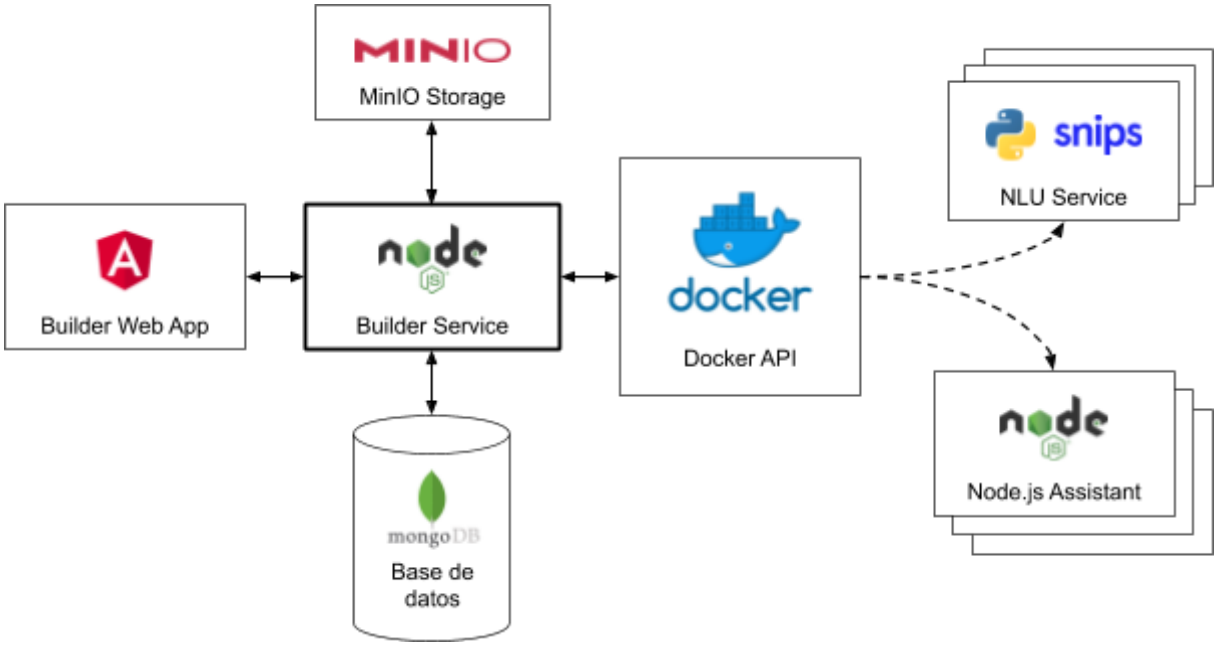


Conectando dos nodos

Para conectar un nodo con otro, debemos crear una condición en el nodo origen. Cuando se crea una condición se muestra un pequeño círculo en el lado derecho del nodo. Para crear una conexión entre dos nodos pinchamos y arrastramos el ratón o el dedo desde el círculo de salida hasta el círculo de entrada del otro nodo.

3.5 Servicio de la aplicación visual

Este servicio (Builder Service) expone una API REST para obtener los recursos (datasets, diálogos y configuraciones) del servicio de almacenamiento (MinIO Storage) y para iniciar los servicios que componen un asistente (NLU Service, Node.js Assistant).



En la siguiente tabla defino todos los endpoints de la API. En las peticiones POST los datos se envían en el cuerpo de la petición en formato JSON.

MÉTODO	ENDPOINT
GET	**/assistant**
	Obtiene todos los identificadores de los asistentes.
GET	**/assistant/:assistantId**
	Obtiene los datos de un asistente.
GET	**/assistant/:assistantId/environment/:environmentId**
	Obtiene datos de un entorno de un asistente.
POST	**/assistant/:assistantId**
	Crea un nuevo asistente o actualiza sus datos si ya existe
POST	**/assistant/:assistantId/environment/:environmentId**
	Crea un entorno de un asistente o actualiza sus datos si ya existe.
GET	**/assistant/:assistantId/environment**
	Devuelve una lista con todos los entornos de un asistente.
DELETE	**/assistant/:assistantId**
	Borra un asistente y sus entornos
GET	**/skill**
	Obtiene una lista con todos los identificadores de los skills
GET	**/skill/:skillId/version**
	Obtiene el ID, nombre, descripción y versiones de un skill
GET	**/skill/:skillId/version/:versionId**
	Obtiene todos los datos de todos los idiomas de la versión de un skill
GET	**/skill/:skillId/version/:versionId/dialog**
	Obtiene todos los datos de todos los idiomas de la versión de un skill (no incluye los datasets)
GET	**/skill/:skillId/version/:versionId/language**
	Obtiene una lista con los códigos de todos los idiomas del skill
GET	**/skill/:skillId/version/:versionId/language/:languageCode**
	Obtiene todos los datos de un idioma de un skill
GET	**/skill/:skillId/version/:versionId/language/:languageCode/dialog**
	Obtiene todos los datos de un idioma de un skill (excepto el dataset)
POST	**/skill/:skillId/version/:versionId**
	Crea o actualiza los datos de la versión de un skill
DELETE	**/skill/:skillId**
	Borra una skill completamente (todas las versiones e idiomas)
DELETE	**/skill/:skillId/version/:versionId**
	Borra una versión de un skill
DELETE	**/skill/:skillId/version/:versionId/language/:languageCode**
	Borra los datos de un idioma de una versión de un skill

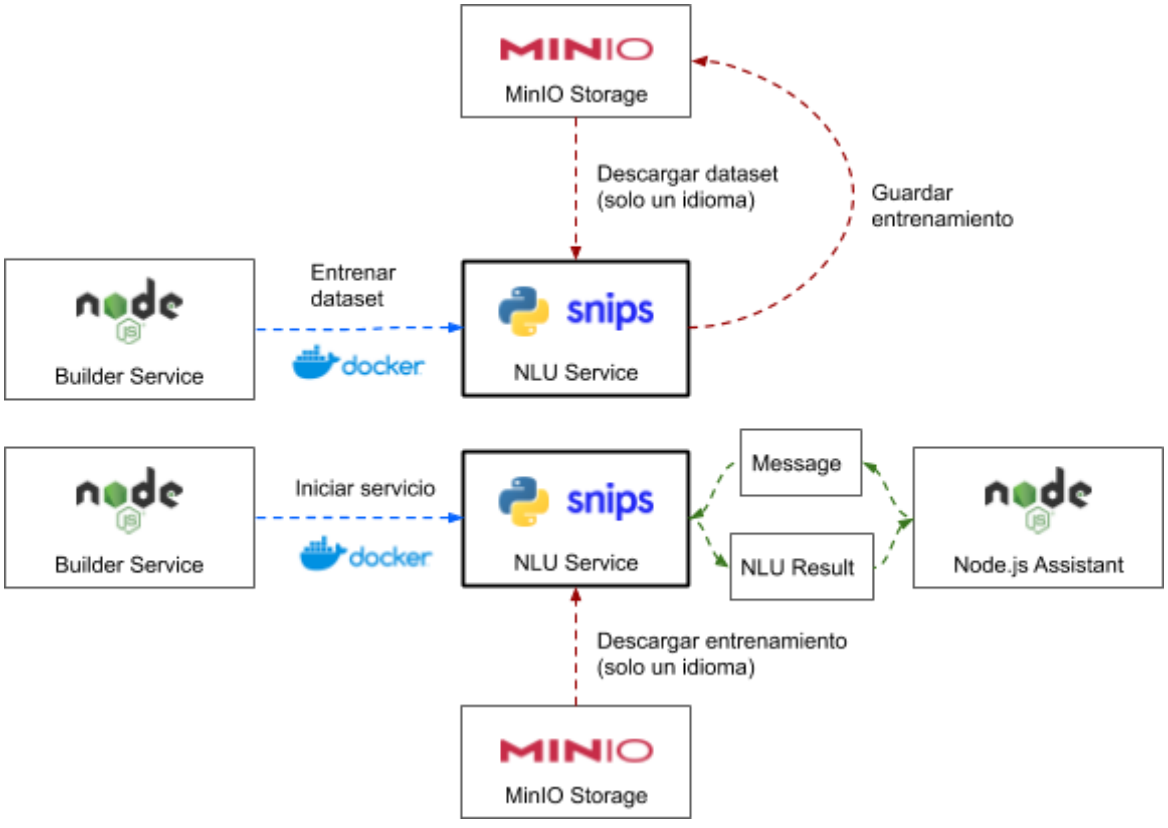
Los siguientes endpoints sirven para iniciar y parar un asistente:

MÉTODO	ENDPOINT
POST	**/assistant/:assistantId/environment/:environmentId/start**
	Inicia los servicios "NLU Service" y "Node.js Assistant" con el asistente y entorno especificados
POST	**/assistant/:assistantId/environment/:environmentId/stop**
	Detiene los servicios "NLU Service" y "Node.js Assistant" del asistente y entorno especificados

Antes de llamar a estos endpoints es necesario que las imágenes de Docker de los servicios “NLU Service” y “Node.js Assistant” están creadas. Este servicio utiliza la API de Docker para crear y ejecutar los contenedores con estas imágenes.

3.6 Servicio NLU

El servicio “Builder Service” inicia este servicio para realizar el entrenamiento con los datasets creados y posteriormente para utilizar este entrenamiento y analizar los mensajes recibidos.



Para realizar el entrenamiento, el contenedor Docker se inicia en modo "entrenamiento" y recibe como parámetros la localización del dataset dentro del servicio de almacenamiento (MinIO Storage) con el que entrenar y la localización donde debe guardar el entrenamiento:

Dataset: *skills/<skillId>/<version>/<languageCode>/dataset*
Entrenamiento: *skills/<skillId>/<version>/<languageCode>/dataset/train.zip*

Una vez que ha terminado el entrenamiento, se puede iniciar el servicio en modo "servicio". En este modo recibe como parámetro la localización del fichero de entrenamiento. En este modo se habilitan dos endpoints para enviar un texto y recibir los resultados del procesado:

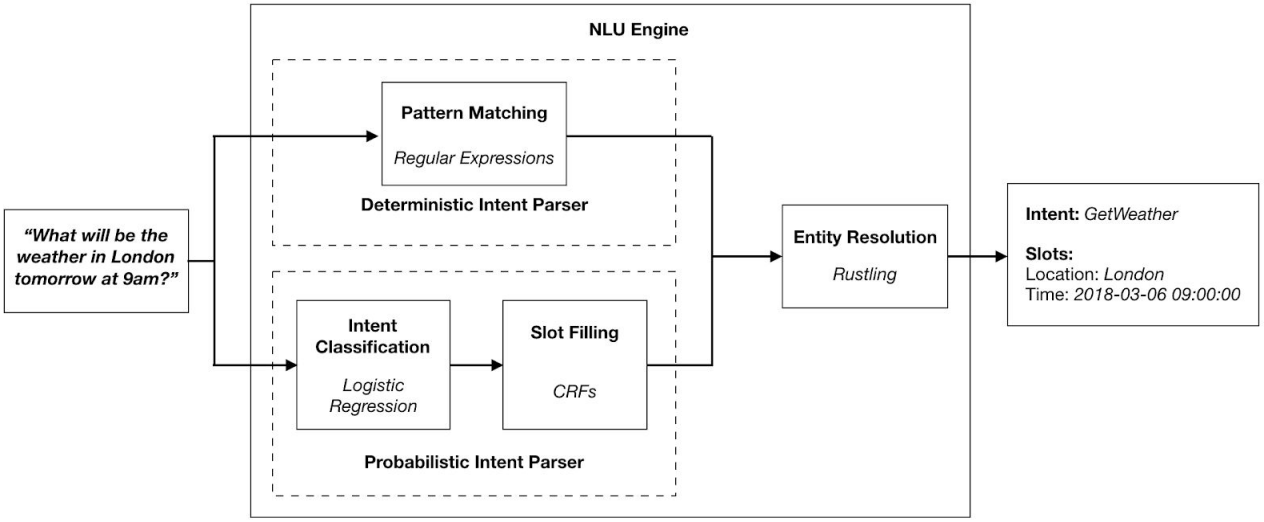
MÉTODO	ENDPOINT
GET	**/nlu?text=<texto>**
POST	**/nlu**

Con el método POST se envía el texto en un JSON. Un ejemplo de respuesta del servicio sería el siguiente:


```
{
   "input": "Weather in Madrid",
   "intent": {
       "intentName": "GetWeather",
       "probability": 0.8408182096904603
   },
   "slots": [{
       "entity": "location",
       "range": {
           "end": 17,
           "start": 11
       },
       "rawValue": "Madrid",
       "slotName": "weatherLocation",
       "value": {
           "kind": "Custom",
           "value": "Madrid"
       }
   }]
}
```

En este ejemplo podemos ver que se ha enviado la frase “weather in Madrid” y ha detectado la intención “GetWeather” y que “Madrid” pertenece a la entidad “location” y al slot “weatherLocation”.

Para obtener estos resultados tan precisos, Snips NLU utiliza la siguiente arquitectura:

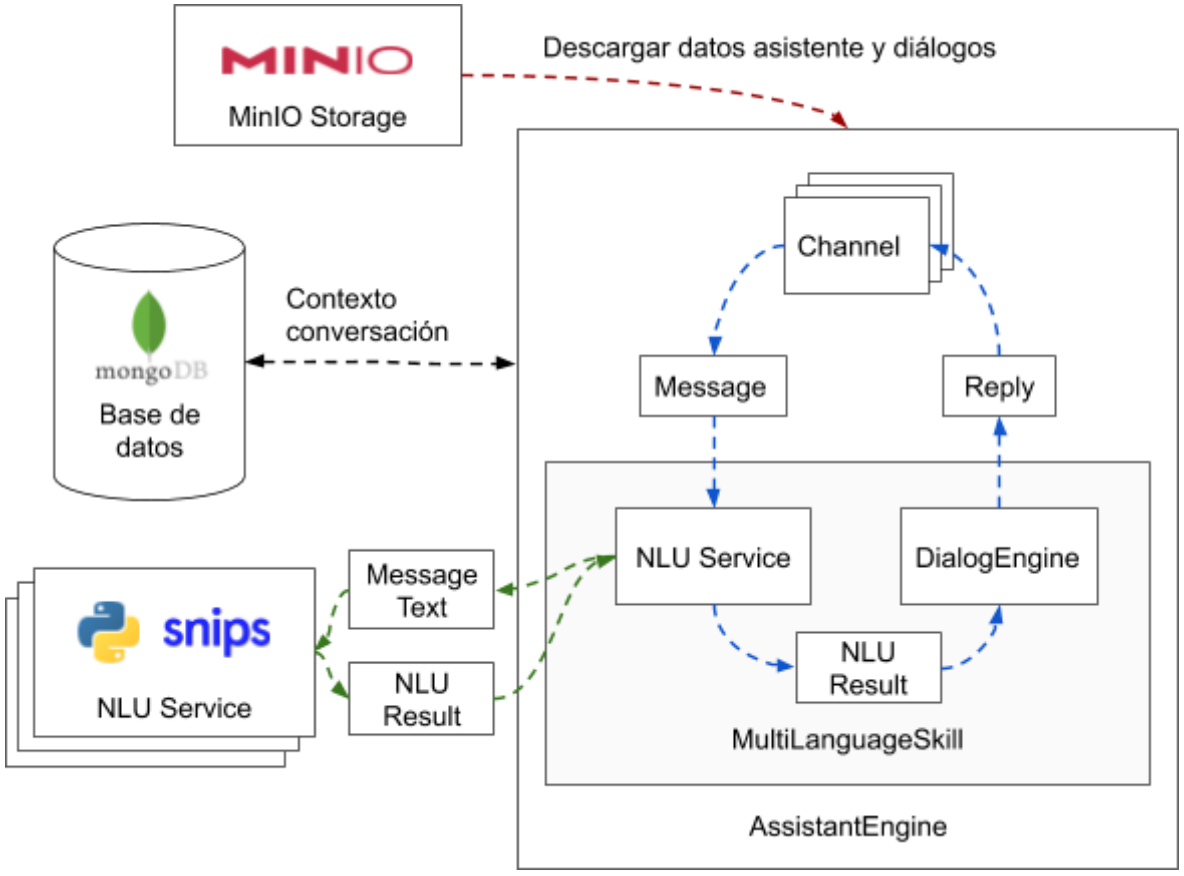


Arquitectura de Snips NLU [63]

Primero utiliza un parseador de intents determinista. Este parseador hace uso de expresiones regulares para identificar intents y slots. Como es el parseador más estricto, es el que primero se usa. El problema que tiene es que no es capaz de generalizar. Si este parseador no consigue encontrar ninguna coincidencia, se utiliza un parseador probabilístico. Este parseador utiliza el aprendizaje automático para ser capaz de generalizar más allá del conjunto de frases de entrenamiento y así poder detectar el intent de frases que nunca ha visto. Para clasificar las frases en intents utiliza la regresión logística, para detectar slots utiliza CRF (Conditional Random Fields) [63]. Después de utilizar cualquiera de los dos parseadores de intents, utiliza la biblioteca Rustling para encontrar entidades como números, ordinales, cantidades de dinero, fechas, intervalos de tiempo o temperatura. Rustling es una biblioteca creada por Snips que recibe ese nombre porque es una reimplementación en el lenguaje Rust de otra biblioteca llamada Duckling [62].

3.7 Servicio de asistente

El servicio “Builder Service” inicia este servicio utilizando como parámetros el identificador del asistente y las direcciones de los servicios NLU, de los Skills que utiliza. Utiliza el identificador del asistente para descargarse los ficheros JSON con la configuración para conectarse a los chats y para descargarse los diálogos. Para cada diálogo crea un motor de diálogo, dependiendo del Skill y el idioma selecciona un motor u otro. El resto del flujo ya ha sido explicado en las secciones anteriores, donde lo nombro como “Node.js Assistant”.



4. RESULTADOS

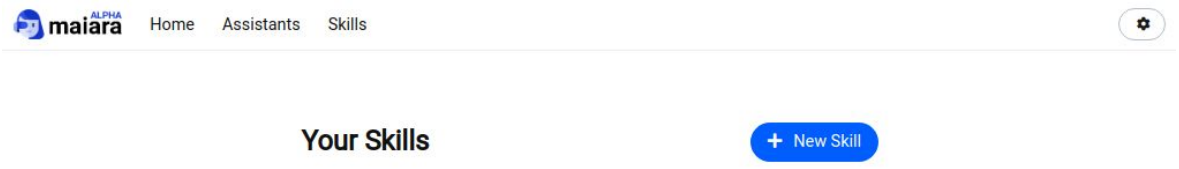
Para iniciar la aplicación vamos al directorio raíz en donde se encuentra el código fuente y ejecutamos el comando “docker-compose up”. Si ejecutamos el comando “docker ps” podemos ver que están iniciados los servicios: la página web (maiara/builder-webapp), el servicio con la API (maiara/builder-service), la base de datos y el servicio de almacenamiento:

```
~ docker ps
CONTAINER ID        IMAGE                                   COMMAND                  CREATED
1ecdc889b8d7        maiara/builder-webapp                   "nginx -g 'daemon of…"   21 seconds ago
3998b5f710dd        maiara/builder-service                  "docker-entrypoint.s…"   22 seconds ago
87d04f67e128        mongo:4.2                               "docker-entrypoint.s…"   3 days ago
2578f80a2f98        minio/minio:RELEASE.2019-08-21T19-40-07Z "/usr/bin/docker-ent…"   3 days ago
~
```

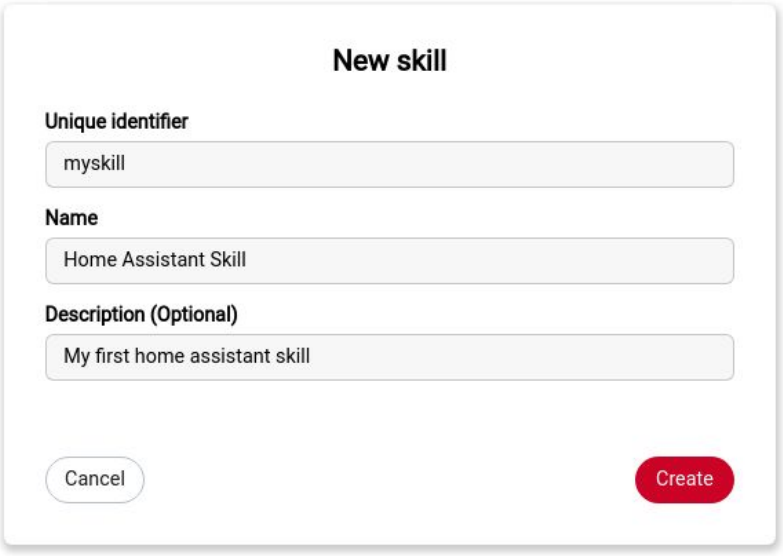
La aplicación web está siendo servida por el puerto 3080. abrimos el navegador y he introducimos la URL “localhost:3080”.

Vamos a crear un asistente para el hogar utilizando un dataset sencillo que Snips NLU tiene de ejemplo. Así podemos comprobar posteriormente que los datos generados con la aplicación son correctos, para ello los ficheros generados deberían ser iguales que los del ejemplo. Con este dataset podemos pedir al asistente el tiempo en una ciudad y que encienda o apague las luces de una habitación.

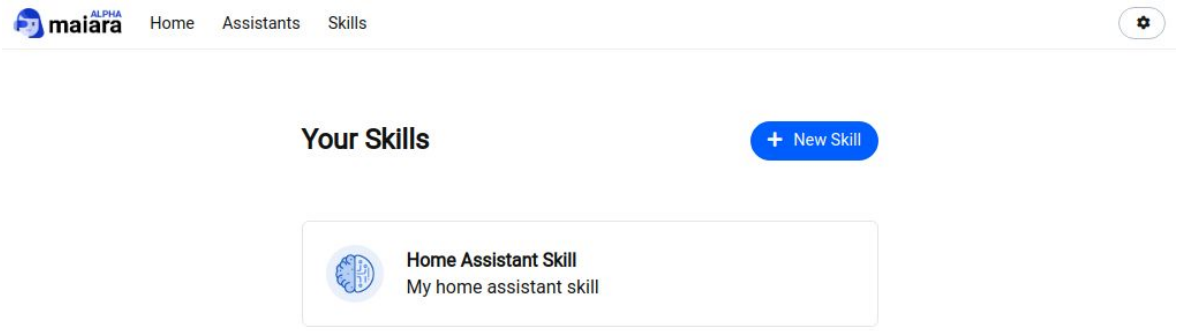
Pinchamos en la opción Skills del menú global y después en el botón “New Skill”



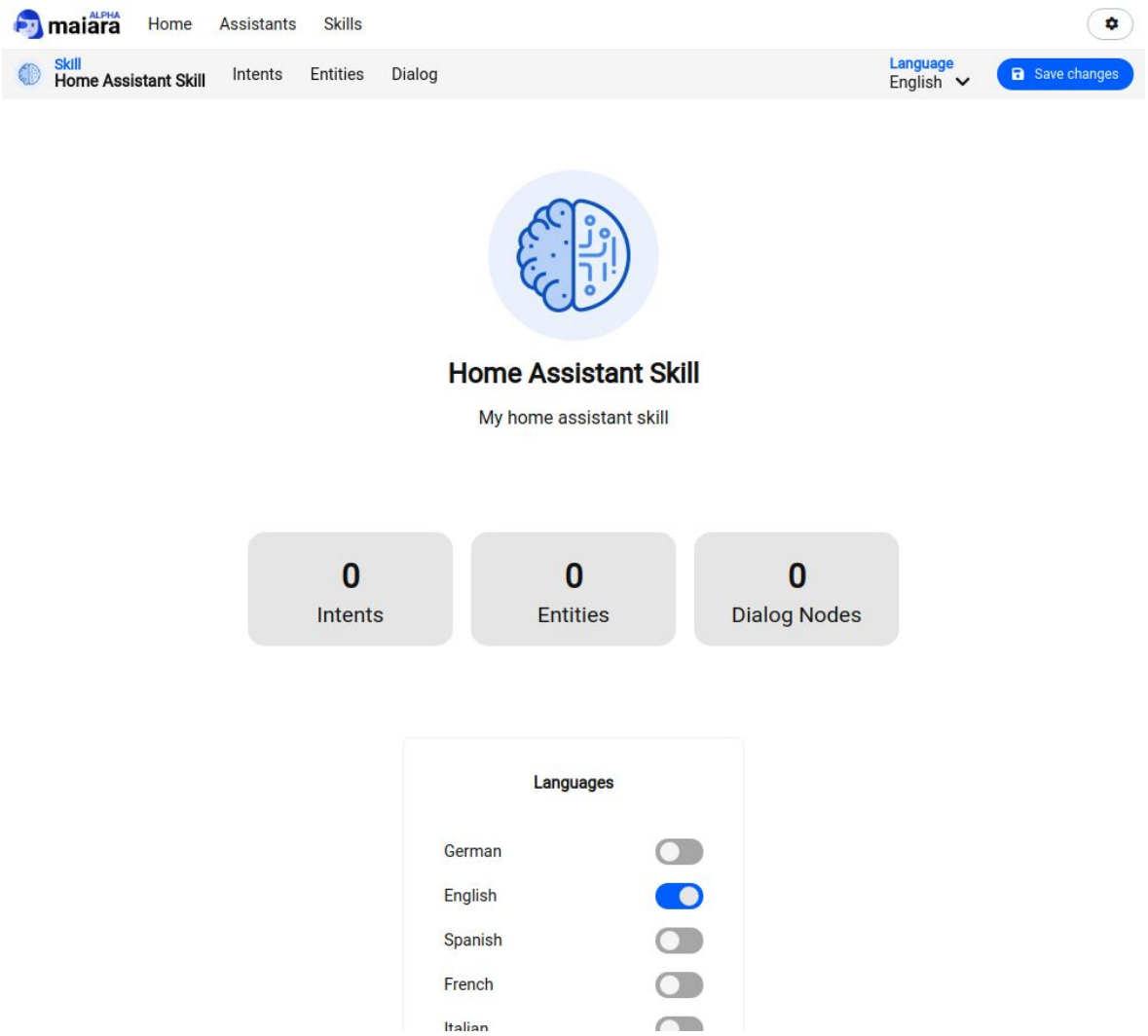
Rellenamos los datos del formulario que aparece y pulsamos “Create”.



Nos aparece la nueva skill en la lista:

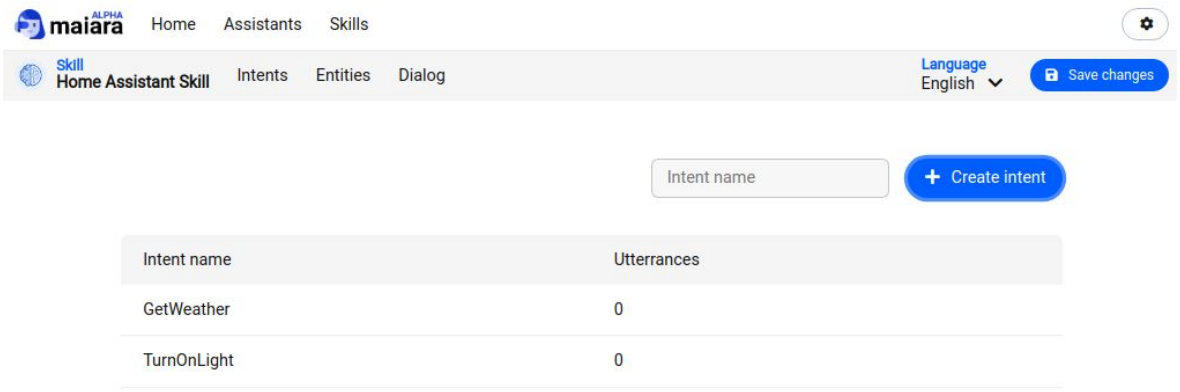


Pinchamos en la nueva skill creada para ir a la página principal de esta skill:

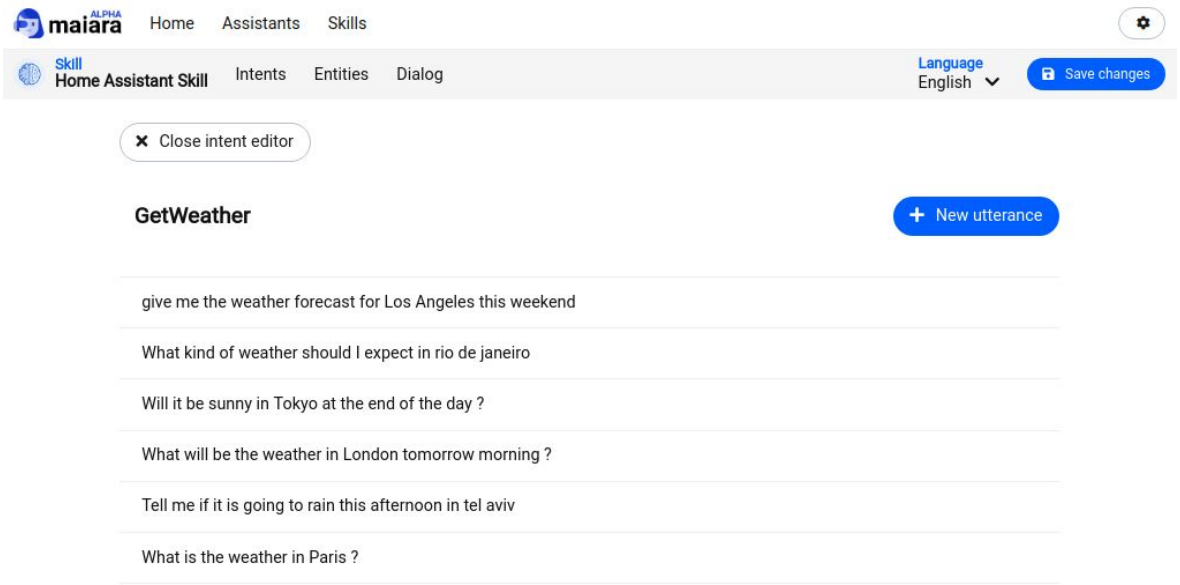


Por defecto se crea con el idioma inglés y sin ningún dato.

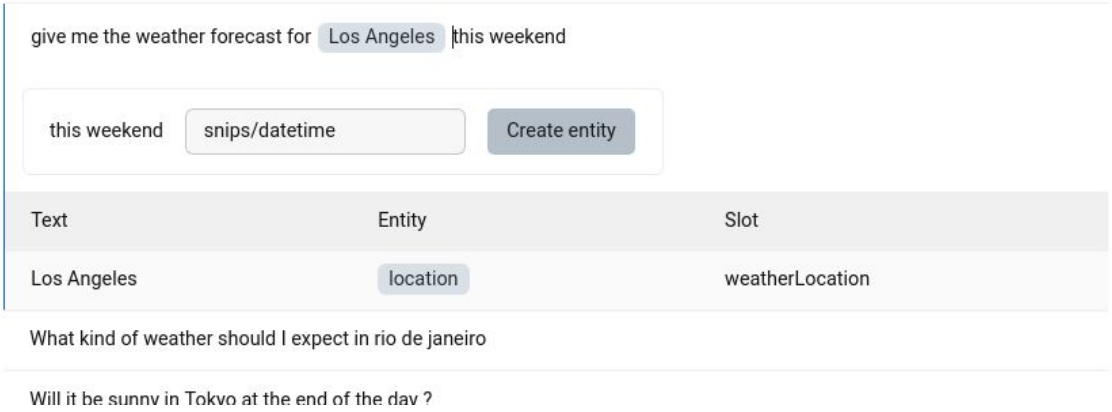
Pulsamos en la pestaña “Intens” de la barra de navegación del asistente. Después introducimos el nombre del intent en la entrada de texto “Intent name” y pulsamos en crear intent. En este ejemplo creamos los intents con nombres “GetWeather” y “TurnOnLight”.



Pinchamos en la fila de la tabla con el intent “GetWeather” para abrir el editor de intents. He introducimos las frases de ejemplo:



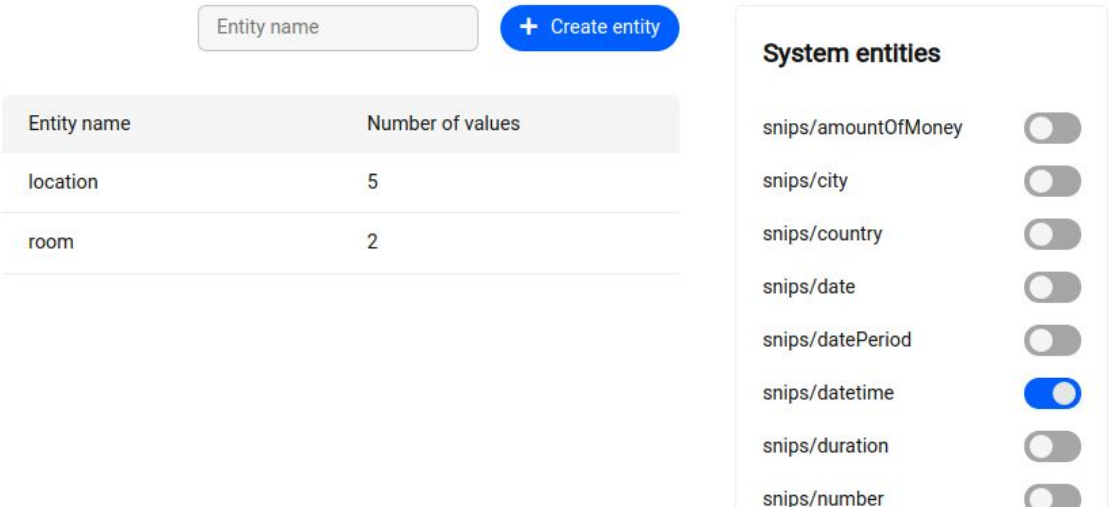
En cada frase marcamos las entidades existentes. En la primera frase marcamos “Los Angeles” con la entidad “location” y el slot “weatherLocation” y “this weekend” con la entidad “snips/datetime” y el slot “weatherDate”.



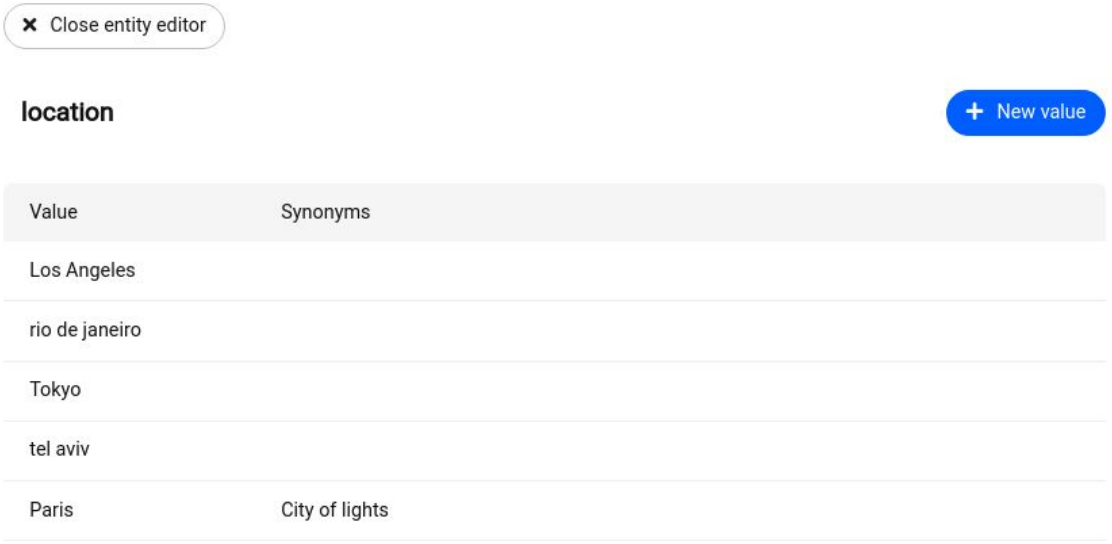
Para marcar una entidad seleccionamos las palabras de la entidad con el ratón. Si no seleccionamos todas las letras de una palabra se selecciona la palabra entera igualmente. Al seleccionar una o varias palabras se abre el formulario para crear una entidad, donde introducimos su nombre. Una vez creada podemos añadir el nombre del slot pulsando en su casilla de la tabla.

Repetimos el mismo proceso pero introduciendo las frases de ejemplo del intent “TurnOnLights”.

Vamos a la página de entidades para comprobar que se han creado las entidades:

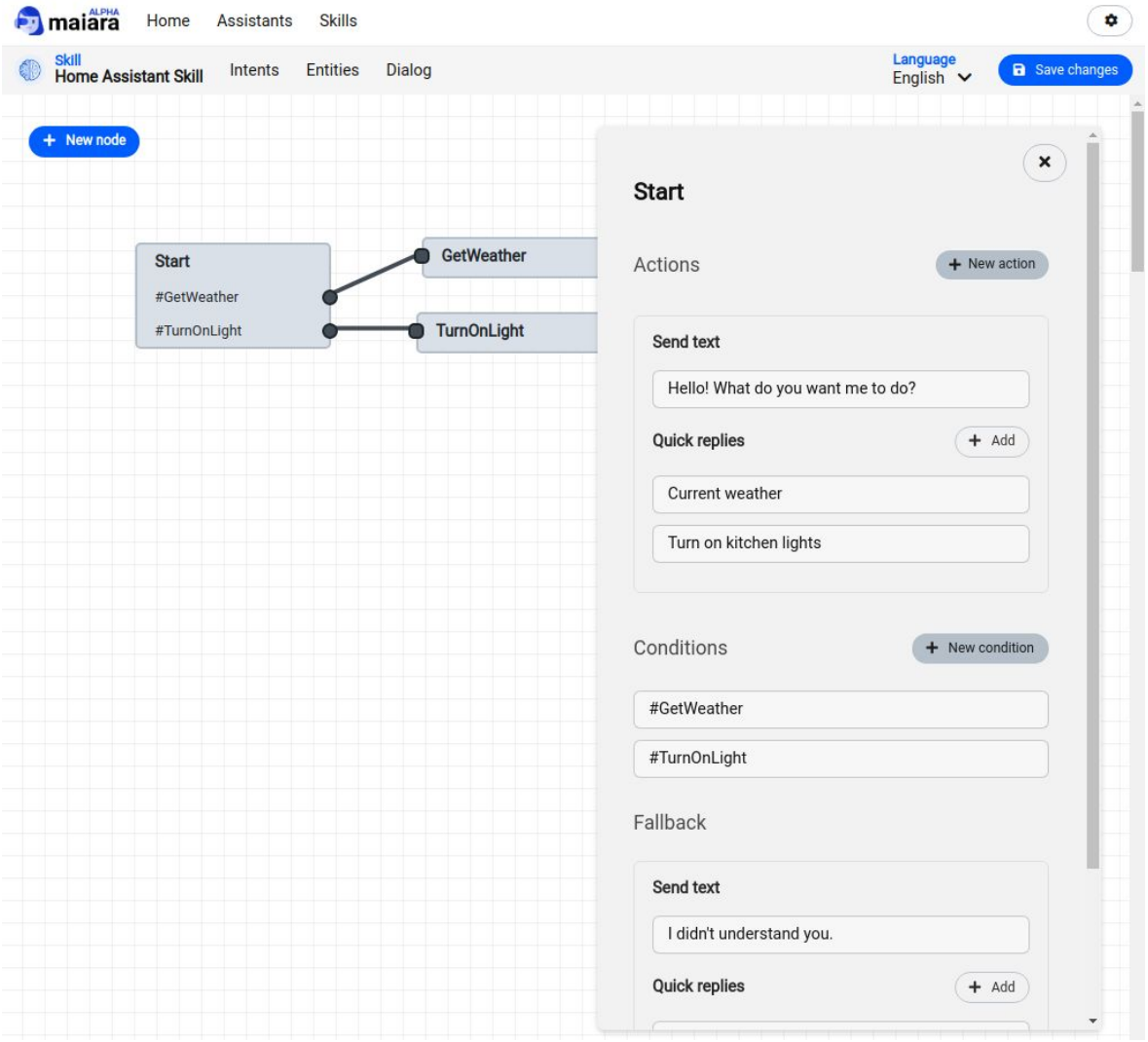


Para editar los valores de una entidad pinchamos en un fila de la tabla y se nos abre el editor de la entidad. Podemos poner como sinónimo de “Paris”, “City of light”.

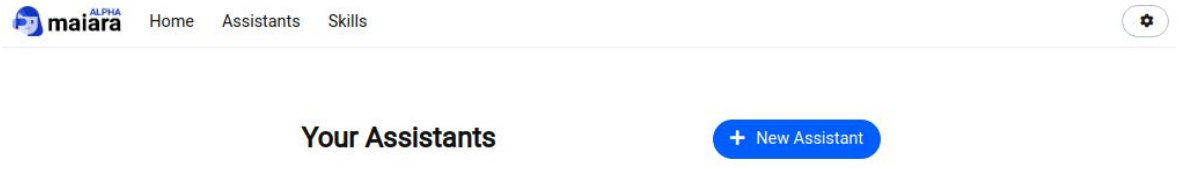


A continuación vamos a la página del editor de diálogo. El nodo “Start” siempre existe. Creamos 2 nodos más pulsando el botón azul de arriba a la izquierda. Pulsamos el nodo “Start” he introducimos una nueva acción de enviar texto. Escribimos como respuesta “Hello! What do you want me to do?” y 2 respuestas rápidas: “Current weather” y “Turn no kitchen lights”. Añadimos 2 condiciones también, en este caso, una por cada intent que hemos creado: “#GetWeather” y “#TurnOnLight”. Después unimos con el ratón el nodo “Start” con los otros nodos. Para saber qué hace cada nodo podemos ponerle a cada uno el mismo nombre que el intent con el que va a entrar en ese nodo.

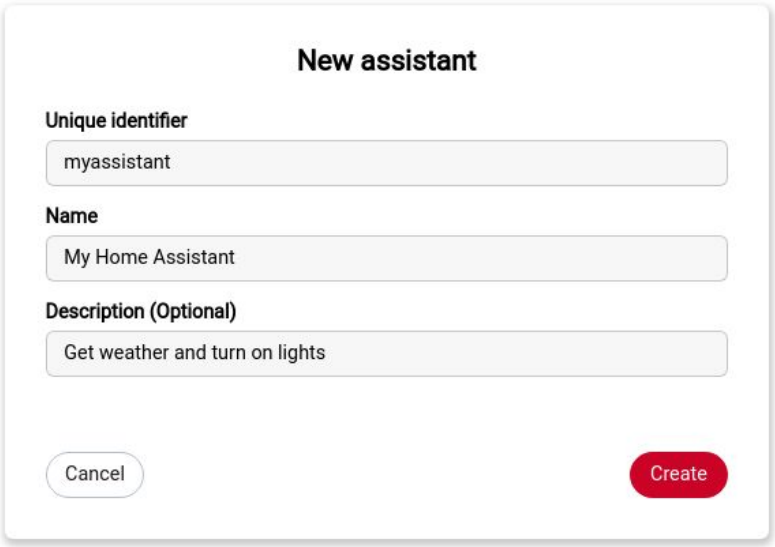
Pulsamos el botón de guardar cambios arriba a la derecha.



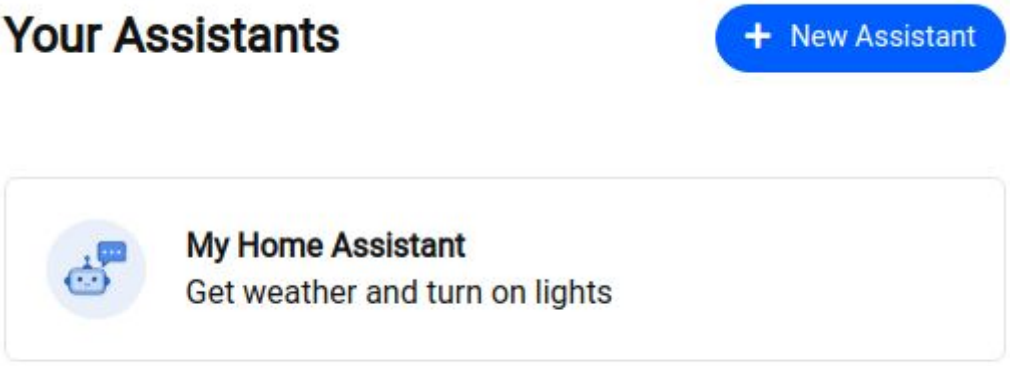
Ahora nos vamos a la página de selección de asistente y creamos uno igual que hicimos con el skill.



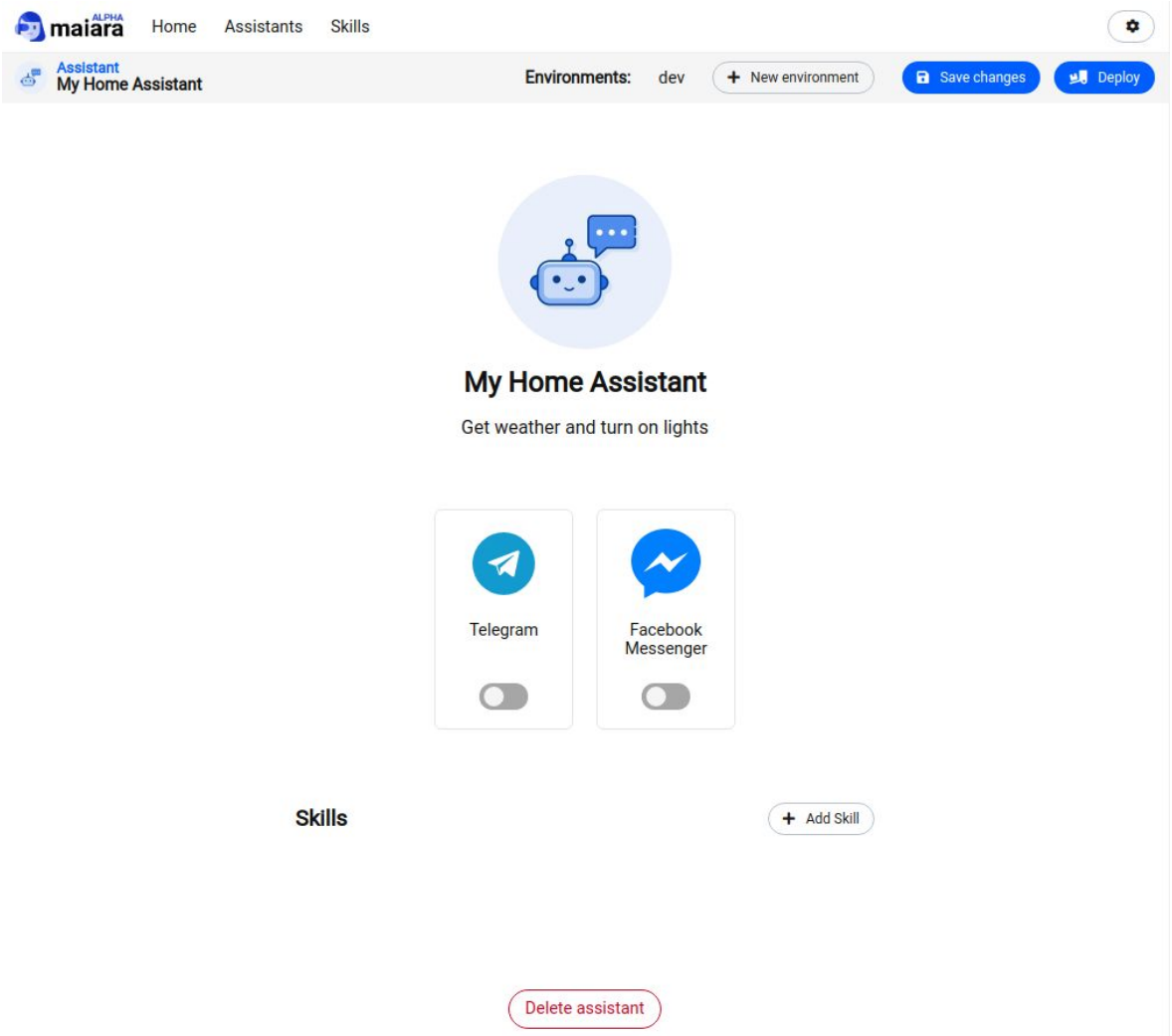
Rellenamos el formulario:



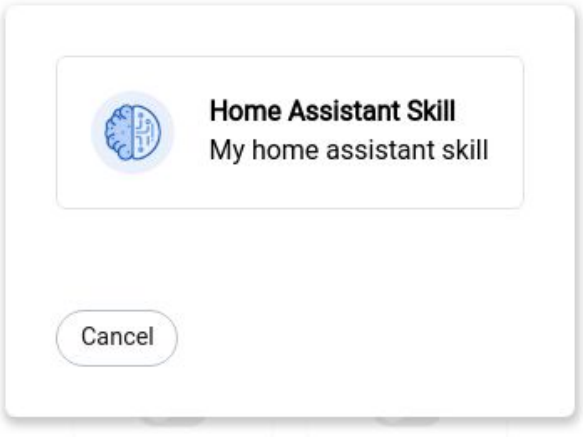
Y se crea el asistente:



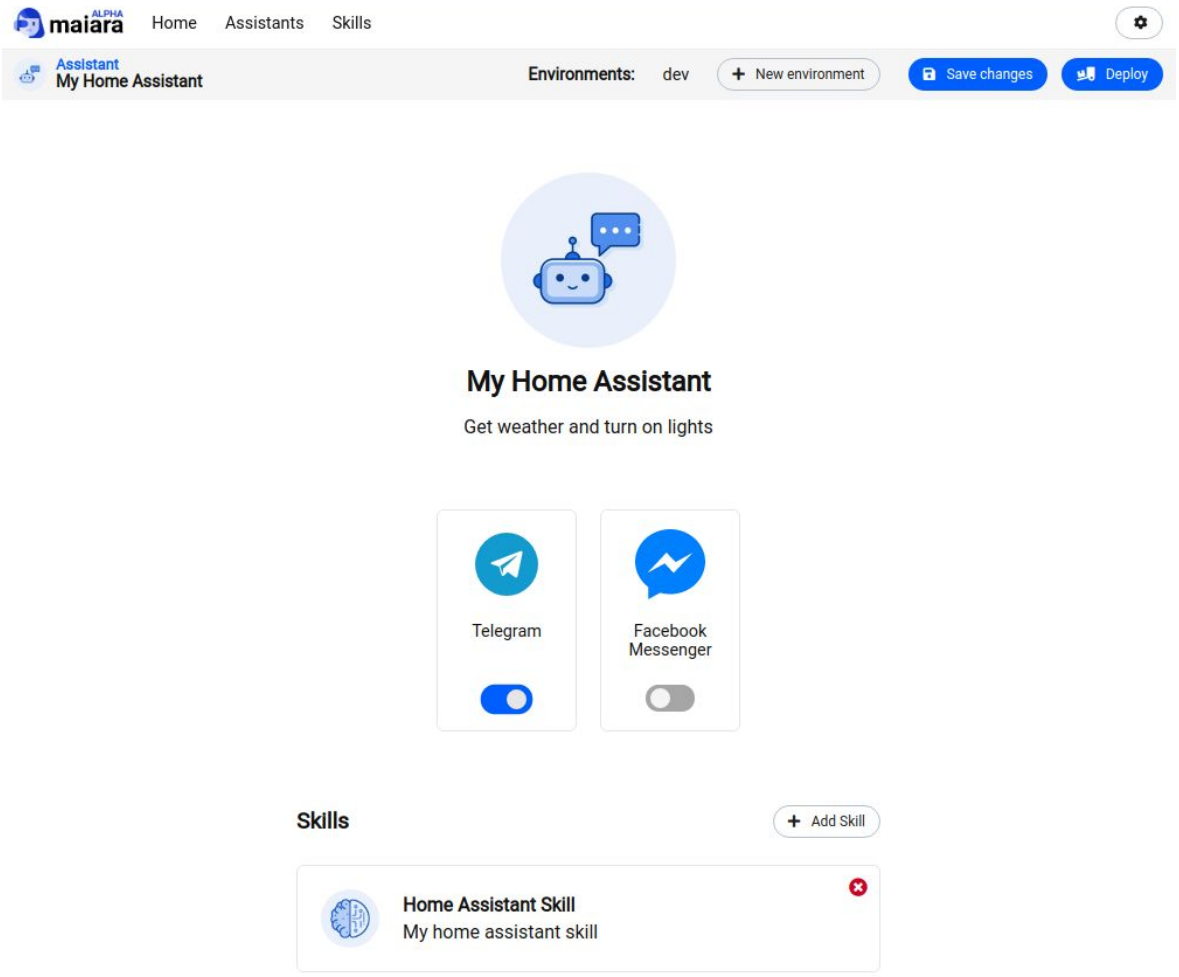
Pinchamos en el asistente creado para ir a la página principal. Después pulsamos el botón “Add Skill”.



Se abre un modal con todos los skills creados, en nuestro caso solo hay uno. Pinchamos en el skill para añadirlo al asistente.

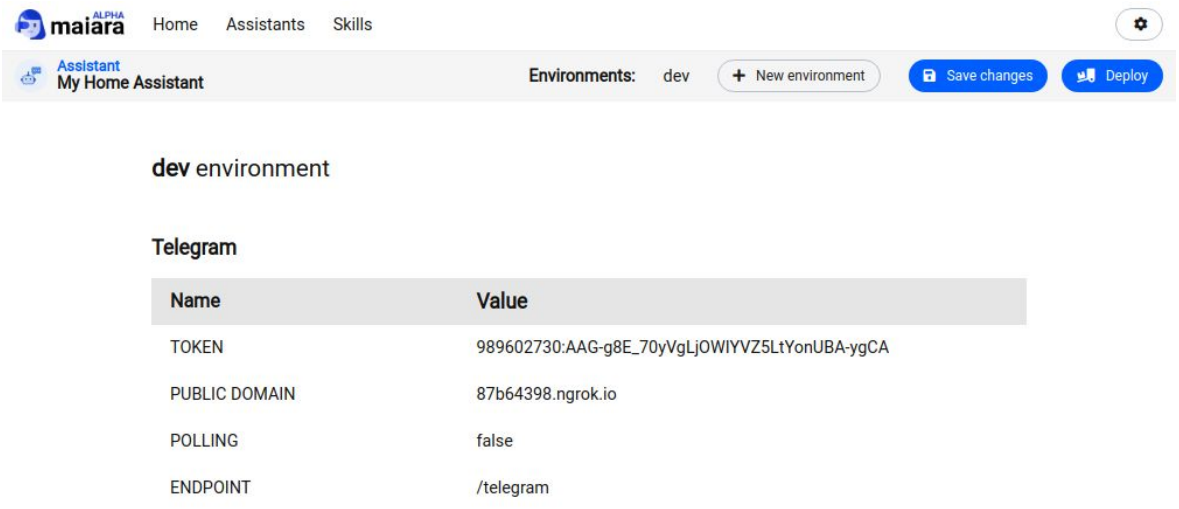


Ahora que tenemos el skill añadido, pulsamos en el botón de tipo “switch”de Telegram para habilitar ese canal.



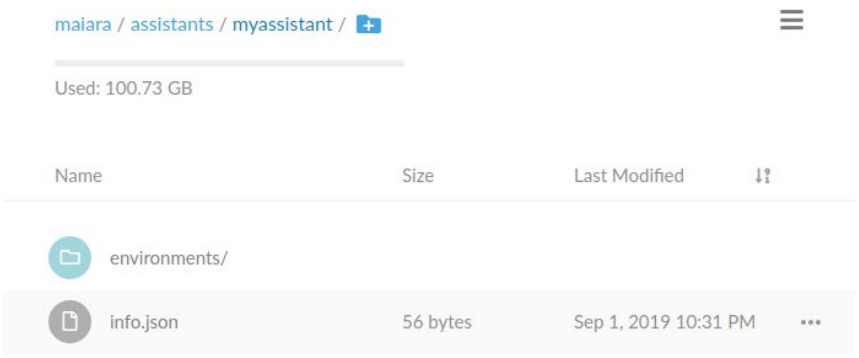
Nos vamos al entorno “dev” creado por defecto y añadimos el token de seguridad de nuestro bot de telegram. Para poder recibir mensajes tenemos dos opciones, poner la opción de POLLING a “true” o conseguir una URL accesible desde Internet. La opción de POLLING es una opción de desarrollo que nos da Telegram y con la que no nos hace falta tener ninguna URL pública. Para conseguir una URL pública podemos hacer uso de aplicaciones como “ngrok” que crean un túnel SSH desde un puerto de nuestro ordenador local a una dominio temporal que nos da la aplicación. Por defecto la aplicación “Node.js Assistant” utiliza el puerto 8080, para obtener una URL escribimos el comando “ngrok http 8080”. Para crear un bot en Telegram y obtener un token tenemos que utilizar el bot de Telegram @BotFather, que nos guiará en el proceso.

Una vez que tenemos todo configurado pulsamos el botón “Guardar cambios” de arriba a la derecha. Ya podemos pulsar en “Desplegar” también. Al pulsar en “Desplegar” se entrena el modelo y después se inicia el asistente.

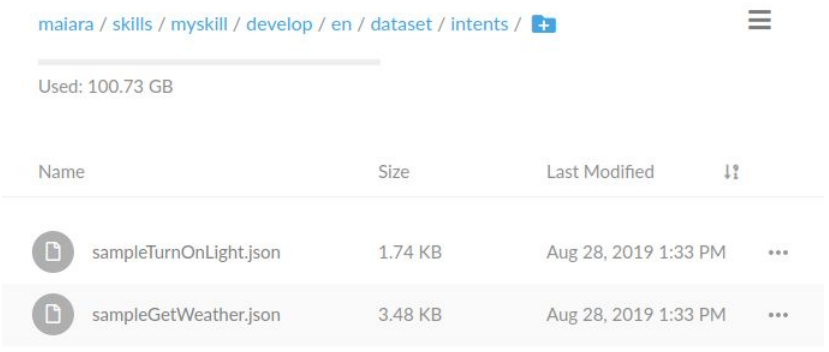


Si abrimos la página de administración de MinIO podemos ver que se han creado todos los ficheros (muestro solo algunas capturas):

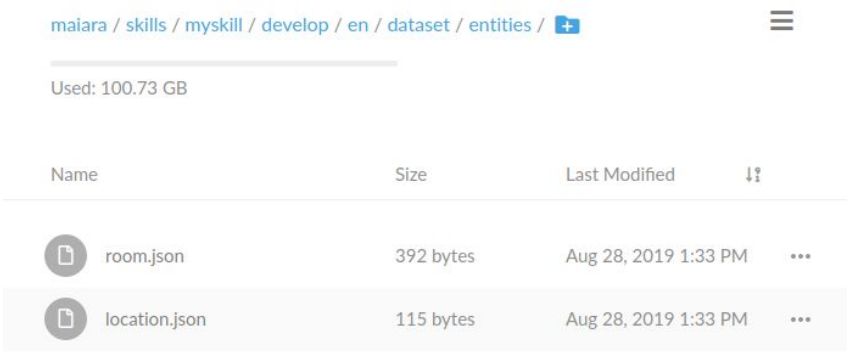
Datos del asistente creado:



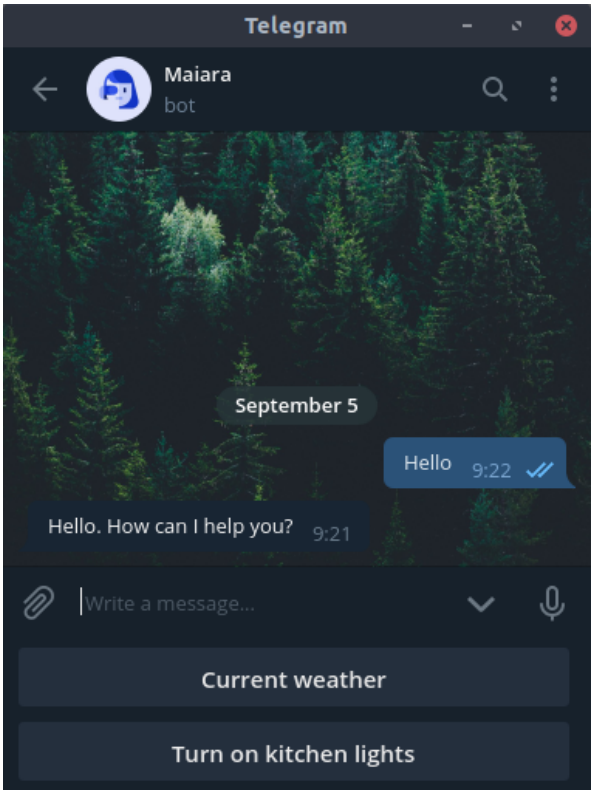
Intents creados en el skill:



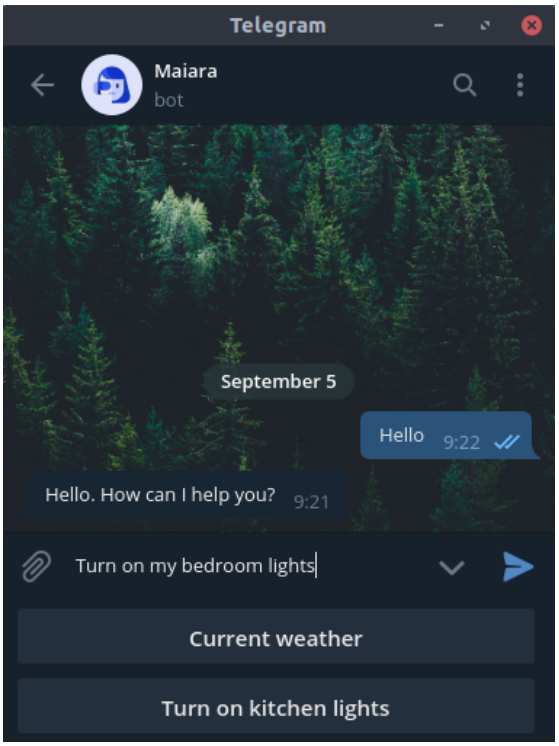
Entidades creadas en el skill:



Abrimos Telegram y buscamos el bot que hemos creado (en mi caso @maiara_bot). Podemos empezar a hablar:



Aunque se muestren los botones de acción rápida se puede escribir lo que se quiera por la entrada de texto normal.



5. CONCLUSIONES Y TRABAJO FUTURO

Antes de comenzar con el diseño hemos repasado las diferentes formas con las que podemos crear chatbots actualmente. Para la aplicación creada hemos utilizado el procesamiento de lenguaje natural (punto 2.3.4) y el gestor de diálogo de estados finitos (punto 2.2.4). Más aplicaciones como IBM Watson o Botpress siguen esta misma metodología. La principal mejora que aporta nuestra aplicación respecto al resto es la posibilidad de utilizar varios idiomas en el mismo chatbot y poder separar el conocimiento de un chatbot en varias skills para intentar tener más modularizado el conocimiento.

Algunas de las mejoras que haría son:

- Terminar de implementar la posibilidad de tener distintas versiones del chatbot. Ahora esta funcionalidad solo se ha semi-implementado en alguno de los servicios creados, pero al ser una tarea difícil de implementar bien, solo permito que haya una versión
- Probar nuevos gestores de diálogo. Uno de los gestores que me resulta más atractivo es Rasa Core. Utiliza aprendizaje automático para predecir las mejores respuestas. De esta forma no depende de reglas que hay que crear a mano. Para este tipo de gestor es necesario tener muchos datos de conversaciones para realizar el entrenamiento y obtener buenos resultados. Si tenemos pocos datos un gestor con reglas sigue siendo una mejor opción.
- Terminar de preparar los servicios para que funcionen en la nube o en ordenadores de placa reducida (Single Board Computer, en inglés), como la Raspberry Pi.

6. REFERENCIAS

1. Ramesh, K.; Ravishankaran, S.; Joshi, A.; Chandrasekaran, K. (2017): A Survey of Design Techniques for Conversational Agents, pp. 336-350. En Kaushik, S.; Gupta, D.; Kharb, L. y Chahal, D. (editores): "Information, Communication and Computing Technology", Springer
2. Canonico, M.; Russis, L. D. (2018). A Comparison and Critique of Natural Language Understanding Tools. Cloud Computing 2018 : The Ninth International Conference on Cloud Computing, GRIDs, and Virtualization.
3. Shane Barker. 13 of the Best AI Chatbot Platforms to Increase Your Conversions. (2018) (Último acceso: 4 de septiembre de 2019) https://chatbotslife.com/13-of-the-best-ai-chatbot-platforms-to-increase-your-conversions-d2ab2c533696
4. Accenture Interactive. Chatbots in Customer Service. (2016) https://www.accenture.com/t00010101T000000__w__/br-pt/_acnmedia/PDF-45/Accenture-Chatbots-Customer-Service.pdf
5. Toni Reid. Everything Alexa Learned in 2018. (Blog oficial de Amazon) (Último acceso: 4 de septiembre de 2019) https://blog.aboutamazon.com/devices/everything-alexa-learned-in-2018
6. Gartner. Gartner Says 25 Percent of Customer Service Operations Will Use Virtual Customer Assistants by 2020. (2018) (Último acceso: 4 de septiembre de 2019) https://www.gartner.com/en/newsroom/press-releases/2018-02-19-gartner-says-25-percent-of-customer-service-operations-will-use-virtual-customer-assistants-by-2020
7. Drift, SurveyMonkey, Audience, Salesforce and myclever. The 2018 State of Chatbots Report. https://www.drift.com/wp-content/uploads/2018/01/2018-state-of-chatbots-report.pdf
8. Botkit (Herramienta de software libre para desarrollo de chatbots). (Último acceso: 4 de septiembre de 2019) https://botkit.ai/
9. Rasa NLU (Herramienta de software libre para procesamiento de lenguaje natural) (Último acceso: 4 de septiembre de 2019) https://rasa.com/
10. Masche, Julia & Le, Nguyen-Thinh. (2018). A Review of Technologies for Conversational Systems. 212-225. 10.1007/978-3-319-61911-8_19.
11. NPR, Edison Research. Smart Audio Report from NPR and Edison Research Spring 2018. https://www.nationalpublicmedia.com/wp-content/uploads/2018/07/Smart-Audio-Report-from-NPR-and-Edison-Research-Spring-2018_Downloadable-PDF.pdf
12. Trips Reddy. How chatbots can help reduce customer service cost by 30%. (Blog de IBM) (2017) (Último acceso: 4 de septiembre de 2019) https://www.ibm.com/blogs/watson/2017/10/how-chatbots-reduce-customer-service-costs-by-30-percent/

13. Juniper Research. Chatbot Conversations to deliver $8 billion in cost savings by 2022. (2017) (Último acceso: 4 de septiembre de 2019) https://www.juniperresearch.com/analystxpress/july-2017/chatbot-conversations-to-deliver-8bn-cost-saving
14. Chen, H.; Liu, X.; Yin, D.; Tang, J. (2018) A Survey on Dialogue Systems: Recent Advances and New Frontiers. arXiv:1711.01731
15. Wikipedia. Chatbot. (Último acceso: 4 de septiembre de 2019) https://en.wikipedia.org/wiki/Chatbot
16. Huang, J.; Zhou, M.; Yang, D. (2007): Extracting Chatbot Knowledge from Online Discussion Forums. Proc. of IJCAI. 423-428.
17. Bradesko, L., & Mladenic, D. (2012). A Survey of Chabot Systems through a Loebner Prize Competition.
18. Turing, A. M. (1950) Computing Machinery and Intelligence. Mind 49: 433-460.
19. Weizenbaum, J. (1966) ELIZA—a computer program for the study of natural language communication between man and machine. Commun. ACM 9, 1 (January 1966), 36-45. DOI=http://dx.doi.org/10.1145/365153.365168
20. Chatbots.org. Definición de chatbot. (Último acceso: 4 de septiembre de 2019) https://www.chatbots.org/chat_bot
21. Chatbots.org A.L.I.C.E. (Último acceso: 4 de septiembre de 2019) https://www.chatbots.org/chatbot/a.l.i.c.e/
22. Pereira, M. J.; Coheur, L.; Fialho, P.; Ribeiro, R. (2016) Chatbots' Greetings to Human-Computer Communication. arXiv:1609.06479
23. Chatbots.org. Albert One. (Último acceso: 4 de septiembre de 2019) https://www.chatbots.org/chatbot/albert_one/
24. Elle Hunt (The Guardian). Tay, Microsoft's AI chatbot, gets a crash course in racism from Twitter. (2016) (Último acceso: 4 de septiembre de 2019) https://www.theguardian.com/technology/2016/mar/24/tay-microsofts-ai-chatbot-gets-a-crash-course-in-racism-from-twitter
25. Wikipedia. Premio Loebner. (Último acceso: 4 de septiembre de 2019) https://es.wikipedia.org/wiki/Premio_Loebner
26. AISB. The Society for the Study of Artificial Intelligence and Simulation of Behaviour - Loebner Price. (Último acceso: 4 de septiembre de 2019) http://www.aisb.org.uk/events/loebner-prize
27. Wikipedia. Mitsuku. (Último acceso: 4 de septiembre de 2019) https://en.wikipedia.org/wiki/Mitsuku
28. Steve Worswick Interview - Loebner 2013 winner. (Último acceso: 4 de septiembre de 2019) https://aidreams.co.uk/forum/index.php?page=Steve_Worswick_Interview_-_Loebner_2013_winner#.XVmVK3UvNhE
29. Pandorabots. Mitsuku. (Último acceso: 4 de septiembre de 2019) https://www.pandorabots.com/mitsuku/
30. Ramesh, Kiran & Ravishankaran, Surya & Joshi, Abhishek & Chandrasekaran, K. (2017). A Survey of Design Techniques for Conversational Agents. 336-350. 10.1007/978-981-10-6544-6_31.
31. Pandorabots Docs. AIML Fundamentals. (Último acceso: 4 de septiembre de 2019) https://pandorabots.com/docs/aiml-fundamentals/
32. AIML Docs (Último acceso: 4 de septiembre de 2019) http://www.aiml.foundation/doc.html
33. AIML 2.0 Reference. (Último acceso: 4 de septiembre de 2019) http://callmom.pandorabots.com/static/reference/
34. Ahmad, S. (2007) Tutorial on Natural Language Processing.

35. Cleverscript. About. (Último acceso: 4 de septiembre de 2019) https://www.cleverscript.com/about/
36. Repositorio de ChatScript en GitHub (Último acceso: 4 de septiembre de 2019) https://github.com/ChatScript/ChatScript/blob/master/WIKI/OVERVIEWS-AND-TUTORIALS/What-is-ChatScript.md
37. Natalya F. Noy and Deborah L. McGuinness (2001) Ontology Development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001.
38. Moyotl-Hernández, E., & Macías-Pérez, M. (2016). Método para autocompletar consultas basado en cadenas de Markov y la ley de Zipf. Research in Computing Science, 115, 157-170.
39. Pérez Ortiz, J. A. (2002) Modelos Predictivos Basados en Redes Neuronales Recurrentes de Tiempo Discreto. Tesis doctoral. Universidad de Alicante.
40. Kathrin. KNIME blog. "Once Upon A Time..." by LSTM Network. (2018) (Último acceso: 4 de septiembre de 2019) https://www.knime.com/blog/text-generation-with-lstm
41. Wikipedia. Redes neuronales convolucionales. (Último acceso: 4 de septiembre de 2019) https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales
42. CS231n Convolutional Neural Networks for Visual Recognition. (Último acceso: 4 de septiembre de 2019) http://cs231n.github.io/convolutional-networks/
43. Colah's Blog. Understanding LSTM Networks. (2015) (Último acceso: 4 de septiembre de 2019) https://colah.github.io/posts/2015-08-Understanding-LSTMs/
44. Sriram, A.; Jun, H.; Satheesh, S.; Coates, A. (2017) Cold Fusion: Training Seq2Seq Models Together with Language Models. arXiv:1708.06426v1
45. Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA. ISBN:0521865719 9780521865715
46. Qiu, M.; Li, F.; Wang, S.; Gao, X.; Chen, Y.; Zhao, W.; Chen, H.; Huang, J.; Chu, W. (2017) AliMe Chat: A Sequence to Sequence and Rerank based Chatbot Engine. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers) 498-503
47. SAS. Procesamiento del lenguaje natural. Qué es y por qué es importante. (Último acceso: 4 de septiembre de 2019) https://www.sas.com/es_ar/insights/analytics/what-is-natural-language-processing-nlp.html
48. SciForce. NLP vs. NLU: from Understanding a Language to Its Processing. (2019) (Último acceso: 4 de septiembre de 2019) https://www.kdnuggets.com/2019/07/nlp-vs-nlu-understanding-language-processing.html
49. Miner, G. Elder, J.; Fast, A.; Hill, T.; Nisbet, R.; Delen, D. (2012) Practical Text Mining and Statistical Analysis for Non-structured Text Data. ISBN: 978-0-12-386979-1
50. Hunter Heidenreich. Introduction to Word Embeddings. (2018) (Último acceso: 4 de septiembre de 2019) http://hunterheidenreich.com/blog/intro-to-word-embeddings/
51. TensorFlow. Vector Representations of Words. (Último acceso: 4 de septiembre de 2019) https://www.tensorflow.org/tutorials/representation/word2vec
52. Landauer, T. K.; Foltz, P. W.; Laham, D. (1998) An introduction to latent semantic analysis. Discourse Processes. Vol. 25. Number 2-3. 259-284 DOI: 10.1080/01638539809545028
53. Bill MacCartney. Understanding Natural Language Understanding. (2014) https://nlp.stanford.edu/~wcmac/papers/20140716-UNLU.pdf

54. Wu, Y.; Wu, W.; Xing, C.; Zhou, M.; Li, Z. (2016) Sequential Matching Network: A New Architecture for Multi-turn Response Selection in Retrieval-Based Chatbots. arXiv:1612.01627v2
55. Braun, D., Hernandez-Mendez, A., Matthes, F., & Langen, M. (2017). Evaluating Natural Language Understanding Services for Conversational Question Answering Systems. SIGDIAL Conference.
56. NLTK (Último acceso: 4 de septiembre de 2019) https://www.nltk.org/
57. WordNet(Último acceso: 4 de septiembre de 2019) https://wordnet.princeton.edu/
58. spaCy (Último acceso: 4 de septiembre de 2019) https://spacy.io/
59. Scikit-learn. (Último acceso: 4 de septiembre de 2019) https://scikit-learn.org/stable/index.html
60. TensorFlow (Último acceso: 4 de septiembre de 2019) https://www.tensorflow.org/
61. Rasa (Último acceso: 4 de septiembre de 2019) https://rasa.com/docs
62. Snips NLU (Último acceso: 4 de septiembre de 2019) https://snips-nlu.readthedocs.io
63. Adrien Ball. An Introduction to Snips NLU, the Open Source Library behind Snips Embedded Voice Platform. (2018) (Último acceso: 4 de septiembre de 2019) https://medium.com/snips-ai/an-introduction-to-snips-nlu-the-open-source-library-behind-snips-embedded-voice-platform-b12b1a60a41a
64. Amazon Lex. (Último acceso: 4 de septiembre de 2019) https://docs.aws.amazon.com/es_es/lex/latest/dg
65. Documentación Dialogflow (Último acceso: 4 de septiembre de 2019) https://cloud.google.com/dialogflow/docs
66. Watson Assistant. (Último acceso: 4 de septiembre de 2019) https://www.ibm.com/cloud/watson-assistant/
67. Microsoft LUIS. (Último acceso: 4 de septiembre de 2019) https://azure.microsoft.com/es-es/services/cognitive-services/language-understanding-intelligent-service/
68. Wit.ai (Último acceso: 4 de septiembre de 2019) https://wit.ai/
69. Google Cloud. Natural Language. (Último acceso: 4 de septiembre de 2019) https://cloud.google.com/natural-language/
70. IBM Watson Natural Language Understanding. (Último acceso: 4 de septiembre de 2019) https://www.ibm.com/watson/services/natural-language-understanding/
71. Amazon Comprehend. (Último acceso: 4 de septiembre de 2019) https://aws.amazon.com/es/comprehend/
72. Botpress (Último acceso: 4 de septiembre de 2019) https://botpress.io/
73. Shane Barker. Chatbots Life. 13 of the Best AI Chatbot Platforms to Increase Your Conversions. (2018) (Último acceso: 4 de septiembre de 2019) https://chatbotslife.com/13-of-the-best-ai-chatbot-platforms-to-increase-your-conversions-d2ab2c533696
74. Data Monsters. Chatbots Journal. 25 Chatbot Platforms: A Comparative Table. (2017) (Último acceso: 4 de septiembre de 2019) https://chatbotsjournal.com/25-chatbot-platforms-a-comparative-table-aeefc932eaff
75. MinIO Docs (Último acceso: 4 de septiembre de 2019) https://docs.min.io
76. Docker (Último acceso: 4 de septiembre de 2019) https://www.docker.com
77. Docker Compose.(Último acceso: 4 de septiembre de 2019) https://docs.docker.com/compose/
78. Kubernetes. (Último acceso: 4 de septiembre de 2019) https://kubernetes.io
79. Google Developers. Web. Progressive Web Apps. (Último acceso: 4 de septiembre de 2019) https://developers.google.com/web/progressive-web-apps/
80. w3schools. HTML Responsive Web Design. (Último acceso: 4 de septiembre de 2019) https://www.w3schools.com/html/html_responsive.asp

81. Wang L, Wang J, Wang M, Li Y, Liang Y, Xu D. (2012) Using Internet Search Engines to Obtain Medical Information: A Comparative Study. J Med Internet Res 2012;14(3):e74
82. Cyc's Knowledge base. (Último acceso: 4 de septiembre de 2019) https://www.cyc.com/archives/service/cyc-knowledge-base